VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Komunikační framework Telepathy a jeho integrace ve Fedoře a KDE

Communication Framework Telepathy and its Integration in Fedora and the KDE Desktop

2013

Martin Klapetek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Martin Klapetek**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Komunikační framework Telepathy a jeho integrace ve Fedoře a KDE

Communication Framework Telepathy and its Integration in Fedora and the KDE Desktop

Zásady pro vypracování:

Telepathy je flexibilní, modulární komunikační framework, který umožňuje různé druhy instantní komunikace díky různým backendům. Cílem práce je zapojit se do aktuálního vývoje frameworku a nového komunikačního prostředku.

Ve své práci proveďte:
1. Popište, jak je Telepathy konkrétně využíváno v desktopových prostředích Gnome a KDE v distribuci Fedora a jaké jsou další možnosti integrace.
2. Prozkoumejte a popište aktuální možnosti tohoto frameworku, případně navrhněte a nebo implementujte vylepšení či chybějící funkce.
3. Zmapujte další možnosti instantní komunikace v prostředí KDE (Kopete, Konversation aj.) a srovnejte s možnostmi Telepathy.
4. V rámci projektu KDE-Telepathy vzniká nový modulární komunikační klient pro KDE. Zapojte se do vývoje následujících částí: contact-list, integrační modul pro pracovní prostředí (kded4 module), plugin pro integrační modul (auto-away system, plugin "práve poslouchám"), integrace Telepathy s frameworkem Nepomuk.
5. Svou práci popište, otestujte a zhodnoťte.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Gaura**

Datum zadání: 18.11.2011

Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2013 .....

Rád bych zde vyjádřil své poděkování, které patří především kolegům z mého týmu - David Edmundson, Daniele E. Domenichelli, George Kiagiadakis, Dan Vrátil a George Goldberg - bez kterých by celý tento projekt nikdy nevznikl. Nezměrnou zásluhu na projektu má také organizace KDE e.V., která projekt zajišťuje finančně. Další poděkování patří vývojářům projektu Empathy, kteří trpělivě odpovídali na všechny mé dotazy o jejich projektu. A v neposlední řadě také Jaroslavu Řezníkovi a firmě Red Hat, kteří mi tuto diplomovou práci umožňili a Janu Gaurovi, který diplomovou práci vedl.

Abstrakt

Tato práce popisuje framework pro instantní komunikaci nazvaný Telepathy a následnou implementaci klienta na tomto framworku založeném. Práce vysvětluje hlavní myšlenku Telepathy „komunikace jako služba“ a jak je tato myšlenka realizována. Druhá část práce popisuje projekt KDE Telepathy, který je kolekcí komponent založených na Telepathy a technologiích KDE a poskytuje integraci instantní komunikace do pracovního prostředí KDE Plasma. Jednotlivé komponenty jsou popsány a vysvětlen způsob jejich implementace. Pro srovnání je vysvětlena také architektura projektu Empathy, což je Telepathy klient pro prostředí GNOME.

Klíčová slova: Telepathy, instantní komunikace, KDE

Abstract

This thesis describes a framework for instant messaging called Telepathy and also an implementation of a client based on this framwork. Thesis explains the main though behind Telepathy „communication as a service“ and how is this thought realized. Next part of the thesis describes the KDE Telepathy project, which is a collection of components based on Telepathy and KDE technologies and provides instant messaging integration into KDE Plasma Workspaces. Each component is described and implementation details are explained. The architecture of Empathy project, which is a Telepathy client for GNOME workspace, is explained for comparison.

Keywords: Telepathy, instant messaging, KDE

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
ABI	–	Application Binary Interface
ofdT	–	org.freedesktop.Telepathy
XMPP	–	Extensible Messaging and Presence Protocol
SASL	–	Simple Authentication and Security Layer
TLS	–	Transport Layer Security
QML	–	Qt Meta Language

Obsah

1 Úvod

KDE je celosvětová komunita vývojářů, umělců, překladatelů a jiných přispěvatelů, jejichž společným cílem je poskytnout alternativu k jiným operačním systémům v podobě svobodného software, který je přístupný všem bez výjimky, zdarma a s plně dostupným zdrojovým kódem. Komunita je zcela otevřená a plně inkluzivní - připojit se a „přiložit ruku k dílu“ může opravdu kdokoliv.

Software vyvíjený a poskytovaný KDE pokrývá širokou škálu aplikací a potřeb uživatele. Součástí aplikací, které KDE v pravidelných půlročních intervalech vydává, je také klient pro instantní komunikaci zvaný Kopete. Na dnešní dobu je ale už bohužel poněkud zastaralý, složitý, neodpovídá moderním trendům použitelnosti a jeho kód je komplikovaný a přináší velké nároky na údržbu. Stav tohoto klienta je dán hlavně faktem, že jde o 10 let starý návrh a často i kód. S příchodem čtvrté generace KDE softwaru, založeného na Qt 4, bylo Kopete velmi pomalu portováno na nové technologie a vývoj vesměs přešel do udržovacího módu. Zdržení zapříčinilo hlavně čekání na nový framework Decibel (kapitola 3.1), na který mělo Kopete být portováno. Skutečné alternativy založené na KDE technologiích neexistovaly a po několik let tak instantní komunikace v KDE softwaru byla spíše zátěží než přínosem.

Jakožto nadšený uživatel jak instantní komunikace, tak KDE softwaru, jsem se stavem instantní komunikace v prostředí KDE rozhodně spokojen nebyl. Naději přinesl nový projekt KDE Telepathy, který si za cíl kladl moderního, rychlého, jednoduchého a velmi dobře použitelného klienta, založeného na technologiích KDE a Telepathy. Protože jsem chtěl změnu a protože byl projekt stejně jako všechen KDE software plně otevřený a svobodný, naskytla se mi jedinečná příležitost podílet se na formování budoucnosti instantní komunikace v prostředích KDE. A proto jsem se do projektu plně zapojil.

2 Telepathy

Aktuálně ve světě existuje několik různých sítí instantní komunikace (dále jen IM), navzájem většinou nekompatibilních s výjimkou protokolů založených na XMPP. Každá síť má navíc svého vlastního klienta, přičemž každý takový klient vypadá jinak, funguje jinak a každý má různé funkce. Jsme-li uživateli více IM sítí, komunikace v různých aplikacích je neefektivní, mnohdy matoucí („je Jarda na ICQ nebo GTalku?“) a také více náročná na zdroje počítače (je nutné mít spuštěno více aplikací).

Tyto a další problémy vedly k volání po jednom univerzálním klientovi, když už ne po jednotné komunikační síti (sjednocení všech uzavřených a licenčně zatížených sítí je prostě utopie). A tak začaly vznikat nové komunikátory, které se snažily tento problém řešit. Ze světa Linuxu jsou známy např. Pidgin či Kopete, z prostředí Windows pak můžeme zmínit třeba Miranda IM nebo Trillian. Obrovskou výhodou těchto klientů je možnost spravovat a především pak používat libovolné množství IM sítí najednou, přičemž zůstává zachován velmi důležitý faktor - konzistentnost uživatelského rozhraní. Uživateli už je pak úplně jedno, je-li jeho kamarád na síti ICQ či Facebooku, všechny své IM kontakty má „sesypány“ do jednoho seznamu v jednom okně.

Na tomto principu vznikl projekt Telepathy, který navíc klade velmi silný důraz na oddělení uživatelského prostředí od implementace protokolů a vlastní logiky řízení. Silná izolace a abstrakce je i mezi vnitřními komponentami. Celý framework je navržen velmi modulárně - každá část je naprosto nezávislá a je možné ji vyměnit za jinou, poskytující stejné nebo podobné služby. Více o architektuře v další kapitole. Aktuálně Telepathy podporuje textovou, hlasovou, video a komunikaci v reálném čase, zároveň umí posílat i soubory a také libovolný proud dat mezi dvěma kontakty pomocí tzv. „tubes“.

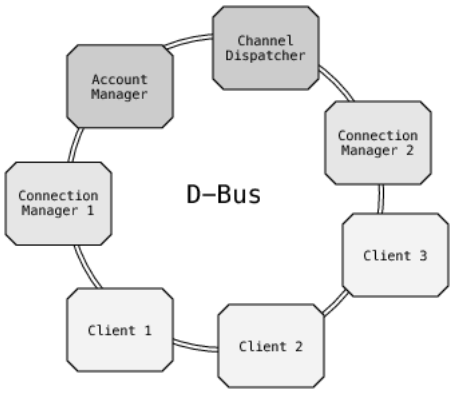
Telepathy je open-source projekt, vyvíjen pod vedením britské společnosti Collabora. Hlavním klientem Collabory pro Telepathy byla Nokia, která framework používala jako základ všech instantních komunikací (včetně GSM hovorů a SMS) na svých telefonech Nokia 770, N800, N810, N900 a N9 včetně vývojářské edice N950. Nokia ovšem od této řady telefonů v roce 2012 upustila s přechodem na operační systém Microsoftu WP7. Projekt Telepathy tedy z pohledu Collabory začal ztrácet na významu a dnes je již spíše v režimu údržby než pod aktivním vývojem.

2.1 Architektura

Telepathy odstíňuje detaily jednotlivých IM sítí pomocí abstrakce - tedy všechny implementace protokolů poskytují stejné API. To samé ale např. dělá i projekt Purple, na kterém je postaven již zmíněný klient Pidgin. Co dělá Telepathy výjimečným, je myšlenka „komunikace jako služba“ - podobně jako např. tisk je systémová služba dostupná všem aplikacím. Poskytování komunikace jako služby má mnoho výhod a zajímavých využití, protože je dostupná celému systému, ne pouze jedné aplikaci. Díky tomu tak můžeme vidět aktuální stav našich kontaktů např. v e-mailovém klientovi, ze kterého můžeme také rovnou zahájit komunikaci bez nutnosti nejprve otevřít náš oblíbený IM komunikátor. Možnosti využití jsou takřka neomezené.

Tuto funkčnost zajišťuje D-Bus a již zmíněný modulární návrh. Díky tomu je bohužel velmi složité přenést framework na jiné systémy, kde D-Bus není dostupný (např. Android nebo Windows). Jednotlivé komponenty spolu komunikují právě přes D-Bus (využívající uživatelskou sběrnici), kde každá komponenta má svou pevně danou cestu, jako např. `/org/freedesktop/Telepathy/AccountManager`. Rozhraní a metody, signály a vlastnosti, které dané rozhraní poskytuje, jsou popsány ve vlastní specifikaci frameworku, která je v době psaní této práce ve verzi 0.27.0 [2].

Na D-Busu jsou 4 základní objekty. **Account Manager** zajišťuje ukládání uživatelem vytvořených účtů a zahájení procesu jejich připojení k síti. Vlastní připojení obstarává **Connection Manager**, což je vlastní implementace protokolu dané sítě. **Channel Dispatcher** pak vyčkává na nové kanály (channels) vytvořené Connection Managerem a deleguje jejich zpracování klientům, kteří hlásí, že daný druh kanálu, kterým může být textový chat, audio nebo video hovor, příchozí soubor apod., jsou schopni zpracovat. Díky této modulárnosti můžeme mít zcela jiné aplikace např. pro chat a video hovory, zcela nezávislé na sobě. Tito klienti tedy zpracovávají vytvořené kanály, ale můžou je také pouze sledovat a do vlastního obsahu kanálu nijak nezasahovat. V aktuální implementaci frameworku jsou Channel Dispatcher a Account Manager v jednom procesu nazvaném **Mission Control.**



Obrázek 1: Jednotlivé komponenty jsou nezávisle na sobě na jedné sběrnici (zdroj: http://www.aosabook.org/en/telepathy.html)

Kromě možnosti použít jinou aplikaci pro každý druh kanálu má tato architektura i další výhody plynoucí z modularity:

- robustnost - selhání jedné komponenty nevede k selhání celého systému (když spadne Account Manager, uživatel může nadále bez problému chatovat)
- jednodušší vývoj - komponenty je možné měnit i za běhu systému za jiné, lze jednoduše testovat více konfigurací komponent
- jazyková nezávislost - protože veškeré řízení procesů probíhá přes D-Bus, nezáleží na programovacím jazyku, ve kterém je vlastní komponenta napsána

- licenční nezávislost - některé licence jsou navzájem nekompatibilní pro běh v jednom procesu, díky modularitě běží vše jako samostatný proces a tento problém tak zcela odpadá
- nezávislost na uživatelském rozhraní - podobně jako jazyková nezávislost, je zcela jedno, jaké uživatelské rozhraní běží nad frameworkem, což umožňuje vytvářet nativní rozhraní pro různá pracovní prostředí jako KDE Plasma, GNOME či mobilní telefony se systémem MeeGo
- bezpečnost - každá komponenta běží ve vlastním adresovém prostoru a ke své činnosti potřebuje minimální privilegia

Jednotlivé komponenty mohou pocházet od různých autorů a jejich jména mohou být úplně neznáma. Aby jednotlivé komponenty o sobě věděly, používá se stejného prefixu v názvu služeb při registraci v D-Busu, který je uveden ve specifikaci frameworku - aktuálně je to `org.freedesktop.Telepathy`, zkráceně `ofdT`. Chceme-li tedy získat např. seznam všech dostupných klientů, stačí v D-Busu vyhledat všechny služby začínající `ofdT.Client`, podobně třeba Connection Managery nalezneme jako `ofdT.ConnectionManager`, existující spojení jako `ofdT.Connection` atd. Celý tento systém je tak plně bezstavový.

2.2 Spojení, kanály a klienti

Spojení (connection) v terminologii Telepathy je vlastní spojení do IM sítě, které iniciuje a spravuje Account Manager, provádí a udržuje jej Connection Manager. Každý připojený účet má jedno vlastní spojení a odpovídá mu jeden objekt v D-Busu.

Kanál (channel) je pak komunikace probíhající po spojení. Může to být tedy textový chat, audio/video hovor, přenos souboru atd., ale také komunikace se serverem nebo ověřování uživatele (tzv. autentizační kanál). Stejně jako spojení, i každý kanál je reprezentován jedním objektem v D-Busu. Kanály otvírá Channel Dispatcher skrz Connection Manager, přičemž je každý kanál svázán se spojením - zanikne-li spojení, zanikne i kanál. Kanály může vytvářet i spojení samotné v reakci na událost, např. příchozí chat, tento kanál je pak předán Channel Dispatcheru, který kanál deleguje klientovi.

Klienti obsluhují nebo pozorují kanály. Telepathy rozlišuje tři druhy klientů

- Pozorující (Observer) - pouze pozoruje kanál bez jakýchkoli zásahů do něj, využívá se např. pro ukládání historie chatů
- Schvalující (Approver) - poskytuje uživateli možnost kanál přijmout nebo odmítnout
- Obsluhující (Handler) - vlastní interakce s kanálem, např. posílání zpráv apod.

Channel Dispatcher deleguje kanály v určeném pořadí - nejprve jsou uvědoměni pozorující klienti a poté je kanál předán schvalujícím klientům. Jakmile jeden ze schvalujících klientů kanál potvrdí, jsou o tom informováni ostatní schvalující klienti a kanál je předán

obsluhujícím klientům, což většinou bývá vlastní uživatelské rozhraní. Klienti mohou v sobě spojovat i více druhů najednou, tedy např. klient může být schvalující i pozorující. Každý klient pak představuje vlastní službu v D-Busu s prefixem „ofdT.Client“. Klienti dále mohou instalovat i soubory, díky kterým je schopen Channel Dispatcher daného klienta automaticky spustit, je-li potřeba. Máme-li nový příchozí chat a není spuštěn žádný klient, který by byl schopen daný kanál obsloužit, podívá se Channel Dispatcher právě na tyto instalované soubory a najde-li klienta, který je schopen daný kanál obsloužit, spustí ho a kanál mu předá. Takový mechanismus má obrovské výhody - klienti nezatěžují zbytečně zdroje daného zařízení tím, že běží, i když nejsou potřeba, a je možné je kdykoli vyměnit za jiné, bez nutnosti dalšího zásahu do jiných komponent.

Výše zmíněná robustnost je jedna z klíčových vlastností Telepathy. Jak Account Manager, tak Channel Dispatcher jsou schopni plně obnovit svůj stav po případném pádu. Při startu se prohledají v D-Busu všechny služby se známým prefixem a všechna existující spojení jsou opět přidružena k existujícím účtům - nenavazuje se tedy nové spojení, pokud je staré aktivní. Podobně i klienti jsou dotazováni na všechny kanály, které právě obsluhují. Pokud spadne klient, který obsluhuje aktivní kanál, Channel Dispatcher automaticky znovu deleguje kanál a klienta restartuje, je-li to potřeba. Pokud klient padá i po novém spuštění, může se pokusit Channel Dispatcher spustit jiného dostupného klienta, není-li žádný dostupný, kanál uzavře. Spadne-li klient obsluhující kanál, ve kterém zůstaly nepřečtené zprávy (každá příchozí zpráva vyžaduje potvrzení, že ji uživatel viděl), tyto zprávy v kanálu zůstanou, dokud je některý klient neoznačí jako přečtené. Příchozí (nepřečtená) komunikace tak zůstává celá zachovaná i po opětovných pádech klienta.

2.3 Connection Manager

Connection Manager je komponenta, která se stará o navazování spojení do IM sítě a následně i veškerou síťovou komunikaci. Implementuje tedy vlastní protokol dané sítě, přičemž jeden Connection Manager může poskytovat podporu i pro více protokolů zároveň. Nad protokolem je pak implementováno rozhraní popsané ve specifikaci Telepathy, díky kterému mohou aplikace využít jakýkoliv protokol bez ohledu na jeho vlastní implementaci a fungování. Takový systém ale zásadně limituje rozšířené možnosti jednotlivých protokolů, protože se v zásadě vytváří průnik funkcionality všech aktivních protokolů. Dalo by se říct, že uživatelé ony rozšířené vlastnosti protokolů stejně nepotřebují a že se lze omezit na pouhý chat, audio a video hovory. Problém ale přichází už v tak základní věci, jakou jsou podporované online stavy jednotlivých protokolů. Více o tom ale v dalších kapitolách.

Z vlastností modularity je velmi vhodné zde zmínit bezpečnost. Connection Managery ke své činnosti potřebují pouze přístup k síti a přístup k D-Busu, a tak je lze velmi jednoduše zařadit do bezpečnostní politiky systému nastavením jasných privilegií a zakázat tak třeba přístup na disk. Každý Connection Manager navíc běží jako samostatný proces, jakékoliv selhání tak opět má pouze minimální dopady na běžící IM systém.

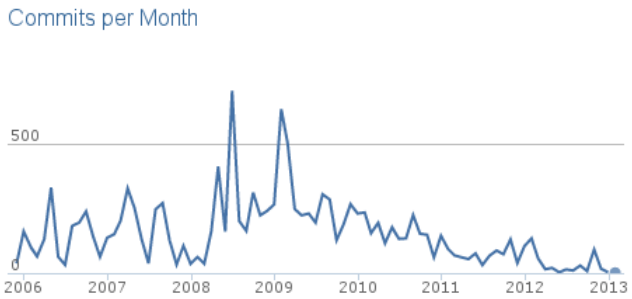
Podobně jako spojení a kanály, i Connection Managery mají v D-Busu prefix `ofdT.ConnectionManager`. A stejně jako klienti, i Connection Manager může být na-

startován, až když ho je opravdu potřeba. Toho je dosaženo instalací speciálních souborů pro D-Bus, které specifikují, co se má provést při žádosti o službu na (zatím) neexistující adrese. Aby Account Manager věděl, jaký Connection Manager pro daný účet použít, název Connection Manageru pro daný účet IM sítě si ukládá. Při připojení účtu se pak Account Manager pokusí získat objekt na prefixu Connection Managera ve spojení s uloženým názvem, např. `ofdT.ConnectionManager.gabble`. Tato adresa v D-Busu zatím neexistuje, protože Connection Manager Gabble ještě nebyl spuštěn. D-Bus díky nainstalovaným souborům služeb ví, který proces spustit a jednoduše jej spustí (Connection Manager se zaregistruje v D-Busu) a Account Manageru pak předá objekt na požadované adrese. Na stejném principu funguje i aktivace klientů Channel Dispatcherem. Celá komunikace je samozřejmě plně asynchronní, aby nedocházelo k blokování jak D-Busu (který je pro bezproblémové fungování Linuxových systémů naprosto zásadní), tak uživatelského rozhraní (což nikdy nedělá dobrý dojem).

Aktuálně je ve vývoji několik Connection Managerů, všechny jsou napsány v jazyce C s použitím knihovny GLib. Knihovna Telepathy-GLib pak poskytuje vše potřebné pro vývoj nových Connection Managerů. Pro vývojáře používající C++/Qt je pak jistě potěšující, že vzniká také podpůrná knihovna pro psaní Connection Managerů s použitím Qt, více v kapitole 3.4

2.3.1 Gabble

Connection Manager Gabble poskytuje připojení do sítí kompatibilních s protokolem XMPP, včetně rozšíření Jingle pro peer-to-peer multimediální komunikaci. Používá se tedy pro připojení do sítí Jabber, Google Talk či Facebook Chat. Gabble podporuje klasický chat, skupinový chat a audio/video hovory. Facebook je poměrně speciální případ, protože pro klienty třetích stran nabízí pouze určitou podmnožinu protokolu XMPP, ve skutečnosti tak Gabble u Facebooku podporuje pouze klasický chat.



Obrázek 2: Měsíční vývoj počtu committů v projektu Gabble (zdroj: http://www.ohloh.net/p/telepathy-gabble)

V době psaní této práce je bohužel vývoj této komponenty na dlouhodobém sestupu s jediným commitem za posledních 30 dní (přičemž v nejaktivnějším období okolo roku

2009 jich v průměru bylo i 250 měsíčně, jak lze vidět na obrázku 2). Na druhou stranu nejnutnější funkce pro IM komunikaci fungují velmi dobře a spolehlivě, pochopitelně tak už není potřeba do projektu investovat velký vývojový čas. Podle sledovací služby Ohloh má Gabble v únoru 2013 zhruba 110 000 řádků čistého kódu (bez komentářů/dokumentace, kterých je dohromady asi 15 000 řádků).

2.3.2 Haze

Telepathy Haze je Connection Manager založený na knihovně libpurple, která ve své aktuální verzi 2.9.0 podporuje 14 protokolů a nespočet dalších jako přídavné moduly třetích stran. Haze začal jako Google Summer of Code projekt v roce 2007.

Díky libpurple tak má Telepathy k dispozici Connection Managera pro všechny známější IM sítě jako jsou ICQ/AIM, Yahoo! Messenger, Gadu-Gadu či MSN. Navíc je libpurple aktivně vyvíjena, protože tvoří základ pro mnoho dalších IM klientů - Pidgin, Adium, Finch, Meebo aj.

2.3.3 Rakia

Connection Manager pro SIP, dříve známý jako Telepathy-SofiaSIP. Rakia (mj. také název pro pálenku původně pocházející z Bulharska/Sofie) pro vlastní SIP komunikaci používá knihovnu Sofia-SIP, jež je implementací protokolu podle RFC3261. SIP je používán pro VoIP komunikaci a je založen na principech HTTP. Díky Rakii je tak možné Telepathy využívat i k VoIP telefonování.

2.3.4 Salut

Salut implementuje link-local XMPP protokol, používaný k bezserverové komunikaci po místní síti, definovaný v XMPP rozšíření XEP-0174. Podle původní implementace společností Apple se protokol také často nazývá Bonjour. Protokol funguje na základě technologií DNS Service Discovery a Multicast DNS. Zjednodušeně řečeno, nový klient na lokální síti oznámí svou dostupnost všem již existujícím klientům, kteří sami udržují seznam okolních aktivních klientů. Navázání spojení pak probíhá výměnou XML zpráv mezi samotnými klienty a vlastní komunikace je v XMPP protokolu [3].

Salut tak umožňuje komunikovat s lidmi na stejné LAN síti bez nutnosti prvotní autorizace kontaktu či přidávání do seznamu kontaktů - každý nový Salut klient na síti se v seznamu kontaktů objeví automaticky.

2.3.5 Ostatní CM

Hned po desktopu jsou hlavním nasazením Telepathy mobilní telefony Nokie, kde je Telepathy využito také pro klasické GSM hovory a SMS/MMS zprávy. Tento Connection Manager, nazvaný Telepathy Ring, je určen především pro systém MeeGo a používá oFono jako svůj backend. Nokia N9/N950 poskytuje i Connection Manager pro Skype - Telepathy Spirit, ten je ale plně proprietární a dostupný pouze na zmíněných Nokiích.

Podpora Skype protokolu v Telepathy je přitom na desktopu jedna z nejžádanějších. Legální podmínky implementace vlastního Skype klienta jsou bohužel v době psaní práce velmi restriktivní, takže je v podstatě nemožné napsat a (legálně) distribuovat cokoli, co by umožňovalo připojení do Skype sítě.

Jedním z dříve vyvíjených Connection Managerů byl i Butterfly, který poskytoval podporu pro síť MSN. Butterfly byl celý napsán v Pythonu, včetně knihovny pro komunikaci se sítí MSN - papyon, což je fork starší a nevyvíjené knihovny pymsn. S verzí 2.5.0 knihovny libpurple se však podpora v Telepathy Haze pro MSN podstatně vylepšila a tím vývoj Butterfly začal ztrácet na smyslu, počátkem roku 2011 pak vývoj zaniká úplně. Oproti Haze podporoval Butterfly také audio a video hovory, specifikace podpory multimediálních hovorů se ale změnila a do Butterfly už se změna nedostala, padl tedy i poslední důvod proč používat/vyvíjet Butterfly.

2.4 Budoucnost

Aktuálně není moc příznivá. Hlavní hnací síla za Telepathy - Nokia - kompletně opustila sféru open-source projektů (Nokia vyvíjela i samotný framework Qt) a mnoho skvělých nápadů zůstalo nedokončených (např. Qt bindings pro snadné psaní Connection Managerů v C++/Qt). Jistou naději na pokračování vývoje dala před pár dny společnost Canonical, která stojí za linuxovou distribucí Ubuntu, svým nejnovějším produktem - Ubuntu pro telefony. Podle všeho bude tento nový systém pro instantní komunikaci používat právě Telepathy a na integraci budou pracovat původní vývojáři.

3 KDE Telepathy

3.1 Historie

Historie projektu sahá až k počátkům KDE4, tedy zhruba do roku 2007. Projekt měl být jedním z hlavních pilířů nového KDE4 a měl sloužit jako základ pro instantní komunikaci. Nesl název Decibel a snažil se o vytvoření KDE API přímo nad Telepathy. Rozhraní, ale i samotná specifikace Telepathy, však procházely velmi bouřlivým vývojem a pro vývojáře Decibelu představovaly časté změny poměrně velký problém. Proto projekt po čase zaniká a v roce 2009 vzniká nový projekt - KDE Telepathy. Tento nový projekt si klade stejný cíl, jako měl Decibel - přinést do KDE jednotnou podporu pro instantní komunikaci; přístup k řešení však byl jiný. Zatímco Decibel přímo implementoval specifikaci, KDE Telepathy používá existující implementaci specifikace spolu s „Qt bindings“ (vrstva nad Telepathy poskytující Qt API). To značně ulehčilo a urychlilo vývoj jak podpůrných knihoven, tak i samotného klienta pro KDE desktop.

Historicky první verze KDE Telepathy - 0.1.0 - byla po letech vývoje vydána 25. července 2011, které značně urychlila má práce. Projekt zpočátku nesl označení „Telepathy-KDE“ a jednotlivé komponenty pak „telepathy-*“, např. „telepathy-contact-list“. Neměl žádné skutečné jméno a s žádným se do budoucna nepočítalo - plán byl začlenit se do pracovního prostředí tak hluboko, že uživatelé budou prostě používat „Chat“. Do poloviny roku 2011 pracovalo na projektu více lidí z Collabory než z KDE. Po první polovině roku se ale situace obrací a projekt je téměř výhradně vyvíjen lidmi z KDE. S narůstající uživatelskou základnou se začalo ukazovat, že projekt potřebuje alespoň jméno pro veřejnou komunikaci s uživateli, ti sami nevěděli, jak sadu aplikací vlastně nazývat. Navíc se v celém projektu mísily různé kombinace pojmenování - „Telepathy-KDE“, „KDE Telepathy“, „KDE IM“, „KTp“ či dokonce „Real-time Communication and Collaboration“. Po vydání verze 0.2 (25. listopadu 2011) bylo navrženo sjednocení všech pojmenování v projektu, včetně repositářů, názvů komponent či interních řetězců v kódu. Zvolen byl název „KDE Telepathy“, jenž měl ukázat, že projekt není řízen společností Collabora jako ostatní komponenty frameworku Telepathy (jež nesly označení „telepathy-*“), ale KDE vývojáři. Pojmenování repositářů (a balíčků) „kde-telepathy-*“ by však bylo příliš dlouhé, proto byla pro názvy používané komponentami zvolena kratší varianta „ktp-*“ (např. „ktp-contact-list“). Byla vytvořena mini specifikace pojmenování, kterou projekt dodržuje [5]. Přejmenování všech částí projektu vyžadovalo pečlivou přípravu a precizní provedení za spolupráce překladatelského týmu (bylo nutné přesunout a přejmenovat existující překlady) a systémových administrátorů (kvůli vlastnímu přesunu repositářů). Změny byly náročné a oblibu týmu zrovna nepozvedly, byly ale potřeba.

3.2 Vize

Vize projektu zůstala stejná jako u Decibelu - jednoduše použitelný framework poskytující možnosti instantní komunikace pro všechny KDE aplikace (kde to má smysl), včetně komunikačních klientů framework přímo využívajících. Původní plán byl vyvinout sadu knihoven pro tvorbu IM aplikací a tuto sadu pak začlenit do kdelibs, což je široká škála

podpůrných knihoven, které tvoří základ jak celého prostředí KDE Plasma, tak i všech KDE aplikací. Projekt kdelibs je založen na frameworku Qt a ve své podstatě se jedná o jeho rozšíření o chybějící funkce, které KDE aplikace potřebují. Počátkem roku 2011 začaly práce na Qt 5, nové generace frameworku, která se snaží o minimální zásah do stávajícího API Qt 4, ale zaměřuje se především na větší modularitu. Krátce po oznámení plánů pro Qt 5 se začalo pracovat na stejné myšlence pro kdelibs - restrukturalizaci všech knihoven k větší modularitě. Chce-li vývojář použít byť jen jednu funkci, kterou kdelibs nabízí, musí do projektu přidat kompletní sadu kdelibs, což je samozřejmě zbytečné či nežádoucí. Byl tedy oznámen nový projekt - KDE Frameworks - a stávající sada kdelibs byla zmražena pro nové funkce. Vize projektu KDE Telepathy se tedy odklonila od plánu s kdelibs a zaměřila se především na vydání vlastního klienta, který měl být plně a hladce integrován do stávajícího pracovního prostředí KDE Plasma. V roce 2012 se pak poprvé začíná uvažovat, že vznikající knihovny by mohly být zařazeny mezi KDE Frameworks, a to odhadem někdy v druhé polovině roku 2013 (po prvním oficiálním vydání KDE Frameworks).

3.3 Architektura

Architektura v mnohém odráží architekturu Telepathy - malé, samostatné a nahraditelné komponenty. KDE Telepathy je tedy sada samostatných aplikací, kde každá plní jeden úkol. Toto striktní rozdělení je aplikováno i na vlastní repositáře s kódem - každá komponenta má tak svou vlastní historii ve verzovacím systému a je vydána jako samostatný balíček. Taková architektura není zrovna optimální pro manuální sestavení celé sady komponent, ale přináší svoje výhody, jako je jednoduchá nahraditelnost komponent. Tyto výhody zatím bohužel ale zůstávají pouze v teoretické rovině, prakticky existují pouze komponenty z projektu KDE Telepathy. S rozdělením nejsou ani příliš spokojení lidé připravující balíčky pro distribuce, protože s každým vydáním musí zpracovat zhruba 15 balíčků, jejichž struktura a počet se zatím mezi každým vydáním změnila. Velmi důležité je, aby distribuční balíčky měly správné závislosti - velmi častou příčinou problémů uživatelských instalací jsou právě některé chybějící balíčky (jako např. komponenta zodpovědná za poskytování hesel).

KDE Telepathy má aktuálně 15 komponent (bez knihovny ktp-common-internals), z toho 11 jich bylo vydáno jako verze 0.6. Zbytek jsou komponenty stále ve vývoji a prozatím nedostatečně stabilní či nedokončené pro vydání.

Zdrojový kód verze 0.6 pro distribuci lze nalézt na `http://download.kde.org/stable/kde-telepathy/0.6.0/src/`. Repositáře všech komponent jsou dostupny na `https://projects.kde.org/projects/extragear/network/telepathy`.

Všechen kód KDE Telepathy je pod licencí LGPL v2.1+.

3.4 Telepathy-Qt

Telepathy-Qt poskytuje vysokoúrovňové Qt API pro práci s Telepathy. Knihovna není pouze vrstvou překládající Qt volání na funkce hlavní knihovny implementované v GLib,

jak tomu velmi často bývá, ale plně samostatnou knihovnou založenou na Qt. Část knihovny je generována ze samotné specifikace Telepathy, která je pro potřeby projektu převedena do XML souborů. Tento vygenerovaný kód poskytuje nízkoúrovňový přístup k D-Bus objektům. Na tomto nízkoúrovňovém základě je pak vystavěn zbytek knihovny s veřejným (vysokoúrovňovým) API. Stejným postupem je vytvářena také knihovna Telepathy-GLib, která poskytuje GLib rozraní pro Telepathy.

Hlavním objektem v nízkoúrovňové části je `Tp::DBusProxy`, který je mapováním D-Bus objektů. `Tp::DBusProxy` je základem pro všechny vygenerované třídy, které přímo pracují s D-Busem. Poskytuje podporu pro volání D-Busových metod, naslouchání D-Busovým signálům, zvládání situací, kdy je D-Busový objekt zneplatněn, D-Bus objekty s volitelným rozhraním a rozšířitelnost.

Volání D-Busových metod je plně asynchronní, aby se neblokovaly aplikace čekáním na odpověď D-Busu, které může být velmi pomalé. Každé volání vrací dočasný `QObject` (`Tp::PendingOperation`), který vyšle patřičný Qt signál, když obdržíme odpověď z D-Busu.

Když zmizí objekt, pro který máme aktivní `Tp::DBusProxy`, z D-Busu, musí samotný `Tp::DBusProxy` objekt zůstat zachován, jeho případné smazání by při dalším použití shodilo celou aplikaci. Proto byl představen koncept zneplatnění objektů. Pokud z jakéhokoliv důvodu objekt z D-Busu zmizí, příslušný `Tp::DBusProxy` objekt je nastaven jako neplatný a všechny metody tohoto objektu selhávají. Přesný důvod, proč byl objekt zneplatněn, lze z objektu získat jako řetězec.

`Tp::DBusProxy` mapuje dva druhy objektů - stavové (stateful) a bezstavové (stateless). Mapování API stavových objektů jako je např. kanál, vždy odpovídá unikátním názvům v D-Busu. Pokud tento název z D-Busu zmizí, je mapovaný objekt automaticky zneplatněný, což odpovídá faktu, že např. daný kanál již neexistuje a nelze jej přímo obnovit do posledního známého stavu (např. když zavěsíme VoIP hovor nebo opustíme chatovací místnost; nový VoIP hovor by vytvořil nový (jiný) kanál). Bezstavové objekty, jako třeba Connection Manager, jsou vázány na známé názvy a pokud objekt s tímto názvem z D-Busu zmizí, mapované objekty zůstávají platné (aplikace poskytující daný D-Bus objekt může být znovu automaticky aktivována a obnovena přesně do předchozího stavu).

Ne všechny objekty stejného typu nutně poskytují stejná D-Bus rozhraní, např. některé objekty `Telepathy.Channel.Type.Text` implementují rozhraní `Telepathy.Channel.Interface.ChatStates`, jiné zase ne. Seznam existujících rozhraní je možné získat přímo z D-Busu voláním metody `Introspect`. To ale není příliš efektivní, protože zpět dostaneme XML, které je třeba zpracovat. Navíc všechny informace v tomto XML obsažené jsou poněkud nadbytečné, stačí nám znát pouze názvy rozhraní, které jsou dostupné, protože při tvorbě samotné knihovny ve statických jazycích už musíme znát dopředu věci jako parametry metod daného rozhraní, jinak knihovnu nezkompilujeme. Pokud pak zavoláme metodu na objektu, který je mapován na D-Bus objekt, který neposkytuje rozhraní s požadovanou metodou, mělo by volání jednoduše skončit s chybou. Telepathy řeší získání dostupných rozhraní vlastností (property) Interfaces na základním rozhraní (tedy pokud chceme zjistit dostupná

volitelná rozhraní pro dané spojení, požádáme D-Bus o vlastnost Interfaces na rozhraní `org.freedesktop.Telepathy.Connection`), jehož jméno známe. To jednoduše vrací seznam řetězců s názvy dostupných rozhraní (pro výše uvedené spojení např. `org.freedesktop.Telepathy.Connection.Interface.ContactList`, `org.freedesktop.Telepathy.Connection.Interface.ContactGroups` atd.). Pokud pak zavoláme metodu, která je z nedostupného rozhraní, je vrácena chyba `Telepathy.Error.NotImplemented`.

Nově vytvořené proxy objekty nemohou být použity okamžitě, musí se nejprve připojit všechny signály a replikovat stav D-Bus objektu (tzn. asynchronní D-Bus volání). Proto byl v Telepathy představen koncept „připravenosti objektů“, který po provedení všech nezbytných operací vyšle signál „připraven“. Po tom, co je objekt připraven, je možné s ním pracovat a lze k jeho vlastnostem přistupovat synchronně (stav D-Bus objektu je replikován lokálně), přičemž stav objektu bude aktualizován signály z D-Busu.

Telepathy-Qt tento koncept dále rozšiřuje o volitelné funkce (features). Ne vždy potřebujeme úplně všechny vlastnosti objektů, proto můžeme vytváření proxy objektů urychlit tím, že specifikujeme, které vlastnosti chceme. Zbylé vlastnosti jsou ignorovány a to včetně D-Bus signálů, odpadá tedy čas strávený jejich zpracováním. Když objekt pak signalizuje, že je připraven, máme zaručeno, že všechny požadované vlastnosti jsou připraveny k použití. Objekty je možné kdykoli rozšířit o další funkce voláním `becomeReady()` a poté čekáním na signál dokončení operace. V tomto je Telepathy-Qt napřed v porovnání s Telepathy-GLib, kde by vývojáři rádi představili stejný koncept [6]. Vlastnosti můžeme buďto nastavit přímo na objektu nebo použít „factories“, které Telepathy-Qt nabízí pro hlavní objekty jako `Tp::Account`, `Tp::Connection`, `Tp::Contact` či `Tp::Channel`. Pro každý takový objekt existuje jedna „factory“, které předáme požadované funkce, a poté tyto „factories“ předáme objektům, které vytváří ty objekty, jejichž funkce požadujeme. Nejčastějším „vstupním bodem“ je `Tp::AccountManager`, neboť ten je hlavní objekt, ze kterého všechny další hlavní objekty přímo či nepřímo pochází (`Tp::AccountManager` vytváří `Tp::Account`, ten vytváří `Tp::Connection`, ten skrze `Tp::ContactManager` vytváří `Tp::Contact` atd.). Vytvořené „factories“ tedy stačí předat objektu `Tp::AccountManager` při jeho vytváření a tím máme zaručeno, že všechny objekty v této hierarchii budou mít požadované funkce.

Telepathy-Qt představuje ještě jeden významný koncept - používání sdílených ukazatelů (Shared Pointers). Tento koncept předchází únikům paměti v podobě ztracených ukazatelů (nesmazaných objektů, na které žádný jiný objekt již nedrží ukazatel; tyto objekty stále zabírají paměť, ale jsou nedostupné). Qt standardně používá mechanismus rodič-potomek, kde každý objekt rozšiřující třídu `QObject` lze přiřadit jako potomek jinému objektu `QObject`. Při smazání libovolného objektu `QObject` se zároveň smaže celá hierarchie jeho potomků. To ale vyžaduje jasnou hierarchii, která v případě Telepathy-Qt buď neexistuje, nebo není možné ji vždy použít.

Proto jsou u všech objektů počítány reference (ref-counting). Je-li počet referencí 0, ještě to ale nezajišťuje automatické smazání objektu. Proto je každý „refcounted“ objekt vložen ihned po svém vytvoření do objektu `SharedPtr`, což je objekt, který sleduje počet referencí vloženého objektu a automaticky objekt smaže, pokud je počet rovný nule. Aby

se zajistilo, že objekt je vždy vložen do objektu `SharedPtr`, mají referencované objekty privátní (popř. chráněný) konstruktor a statickou metodu, která objekty vytváří a vrací již objekt `SharedPtr`.

Všechny objekty (účty, kontakty, Account Manager atp.) se kterými kód KDE Telepathy pracuje, jsou tedy sdílené ukazatele (`Tp::AccountPtr`, `Tp::ContactPtr`, `Tp::AccountManagerPtr`), pouze výjimečně se pracuje s vlastním objektem. Signály pak samozřejmě vysílají vlastní objekty a připojit se je potřeba přímo k nim. Sdílený ukazatel poskytuje přímý přístup k sledovanému objektu prostřednictvím metody `data()`. Připojení k signálům tedy vypadá např. takto:

```
connect(m_account.data(), SIGNAL(iconNameChanged(QString)),
        this, SLOT(updateIcon(QString)));
```

Jak již bylo zmíněno, každé volání metody vyžadující D-Bus volání, např. nastavení avataru účtu, vrací ukazatel na objekt `Tp::PendingOperation`, reprezentující probíhající operaci. Po dokončení operace vyšle objekt signál `finished(Tp::PendingOperation *self)`, ve kterém předává ukazatel na sebe sama (to aby ze slotu, který je k signálu připojen, byla operace přímo dostupná). Ačkoliv u některých operací nemusíme nutně čekat na dokončení (jako např. při nastavení avataru), je vždy dobré k signálu připojit alespoň jednoduchý slot, který ověří, že operace neskončila chybou a pokud ano, na chybu reagovat (např. informováním uživatele o neúspěšné operaci).

Vůbec první vydání knihovny Telepathy-Qt vyšlo v únoru roku 2009, tehdy ještě pod názvem Telepathy-Qt4. Jméno bylo změněno na Telepathy-Qt s verzí 0.9, která už podporuje také Qt5. Během svého života prošla několika nekompatibilními změnami, což si vyžádalo také menší či větší přepis aplikací. Telepathy-Qt4 ve verzi 0.5 je také dodávána na posledním telefonu Nokie se systémem MeeGo - Nokia N9.

Budoucnost Telepathy-Qt je stejně jako budoucnost celého frameworku Telepathy nejistá. Poslední vydání Telepathy-Qt vyšlo 13. července 2012 a od té doby nemá Telepathy-Qt aktivního správce. Tato situace je pro nás nepříznivá, protože jako hlavní uživatelé Telepathy-Qt jsme sami poskytli několik oprav a několik dalších máme připraveno, není ale nikdo, kdo by je začlenil. Podobně zůstalo několik plánovaných funkcí pouze ve stavu plánování. Navíc se našlo pár vývojářů, kteří by rádi dokončili podporu psaní nových Connection Managerů v Qt. Toto je velmi důležitý milník, který jsme připraveni plně podpořit. Telepathy-Qt je naštěstí svobodný software a pokud se situace v blízké době nezmění, máme v plánu projekt převzít a stát se tak aktivními správci.

3.5 Komponenty

3.5.1 Sdílená knihovna ktp-common-internals

Základním prvkem je knihovna ktp-common-internals, která obsahuje sdílené funkce a grafické ovládací prvky, modely (ve smyslu model-view-controller paradigmatu) a utility. Na této knihovně lze velmi snadno vystavět např. nový seznam kontaktů, který bude bez problému fungovat s ostatními komponentami Telepathy. Knihovna nebyla součástí

projektu od počátku, ale vznikla později, kdy s přibývajícími komponentami přibývalo duplikovaného kódu a změna všech těchto částí byla čím dál tím náročnější. První verze fungovala jako git modul - každý repositář, který potřeboval některou ze sdílených funkcí, obsahoval podrepositář „common“, který bylo nutné aktualizovat samostatně. Protože projekt procházel bouřlivým vývojem, byl tento systém nepohodlný a problematický, proto se od něj velmi brzy upustilo a z podrepositáře vznikla samostatná plnohodnotná sdílená knihovna. Protože byla tvořena sbírkou tříd z různých míst a API bylo velmi nestabilní (navíc zcela bez dokumentace), bylo nežádoucí, aby byla využívaná mimo projekt KDE Telepathy. Do kódu, který přidává knihovnu do projektu, byla tak přidána kontrola, zda-li je sestavovaný projekt je „interní “, tedy jedna z komponent KDE Telepathy, pokud ne, cílový projekt není možné sestavit. Kontrola je velmi jednoduchá a lze ji snadno obejít přidáním

```
set(IS_KTP_INTERNAL_MODULE TRUE)
```

do sestavovacího systému projektu. Tím ale vývojář vědomě přijímá to, že API se může každou chvíli změnit a že je pouze na něm, aby udržoval svou aplikaci funkční (a bez pádů).

S plánováním verze 0.6 se už dělají první kroky ke stabilnímu API a na mnoha místech už je potřebná dokumentace. Plně stabilní API (a ABI) se aktuálně plánuje pro verzi 1.0. Také se uvažuje o zařazení verze 1.0 mezi KDE Frameworks, v době psaní práce však ještě neexistují žádné konkrétní plány pro tento krok.

3.5.1.1 Modely Mým prvním úkolem v projektu bylo vytvořit falešný model pro seznam kontaktů, na kterém by se aplikace testovala a vyvíjela. Model-View-Controller paradigma, jak jej popsal Gamma et al. v [4], je v Qt frameworku zjednodušeno pouze na Model-View, přičemž „View“ v sobě kombinuje „View“ a „Controller“. Tento přístup stále plně rozděluje datové struktury a jejich prezentaci uživateli - stejná data lze tedy zobrazit různými způsoby bez nutnosti změn samotných datových struktur. Model-View v Qt je možné rozšířit o „Delegates“, které poskytují programátorům naprosto volnou ruku při vykreslování vzhledu jednotlivých položek pohledu a způsobů úpravy dat modelu. Qt obsahuje několik připravených tříd pro Model-View programování pro jednoduché a rychlé použití a sadu abstraktních tříd pro implementaci vlastních modelů resp. pohledů a delegátů. V případě modelu pro seznam kontaktů se však muselo začít od nuly, protože datová struktura kontaktů je strom (kontakty jsou řazeny do skupin) a Qt pro stromové struktury model nenabízí.

První model byl jednoduchou podtřídou `QAbstractItemModel`, jenž definuje rozhraní, které modely musí implementovat, aby mohly být použity se zbytkem Qt Model-View komponent. Toto rozhraní reprezentuje data jako hierarchickou strukturu, která je tvořena tabulkami dat.

Data z modelů jsou získávány pomocí indexů (`QModelIndex`). Data v samotném modelu tak můžou být uložena naprosto jakkoliv, model musí pouze být schopný převést index na vlastní data. Index tak není nic jiného, než jednoduchá (a dočasná) struktura obsahující pozici požadovaného záznamu a typ požadovaných dat, tzv. role. Při vlastním

vykreslování tedy pohled nejprve zjistí, kolik položek model celkem obsahuje (voláním `QAbstractItemModel::rowCount()`), a poté postupně předá modelu index každé položky. Při použití standardních delegátů se po modelu požadují dva druhy dat - zobrazovací role a dekorační role. První role je text, který se má u položky vykreslit, druhá role je ikona. Použijeme-li vlastní delegáty, máme vykreslování plně pod kontrolou a můžeme model požádat o data pomocí jakýchkoliv rolí chceme (samozřejmě model pro ně musí mít data). Pohled postupně vykresluje jednu položku za druhou, přičemž model pomocí vlastního Qt mechanismu signálů a slotů informuje pohled o veškerých změnách v datech (předáváním změněných indexů), což zaručuje zobrazování vždy aktuálních dat obsažených v modelu.

Datová struktura prototypu modelu byla také jednoduchá - objekty navzájem propojené ukazately na svoje podobjekty. Každý objekt obsahoval seřazený seznam podobjektů, přičemž samotný model obsahoval seřazený seznam všech objektů nejvyšší úrovně. Objekty nebyly skutečně seřazeny, měly ale v seznamu neměnné pořadí. Dále měl každý objekt vlastní datový kontejner, který ukládal vlastní data. V konkrétní implementaci tedy model obsahoval seznam skupin, kde každý objekt skupiny držel ukazatele na všechny kontakty v dané skupině. Mapování indexů bylo díky neměnnému pořadí poměrně jednoduché.

Model se manuálně plnil daty v konstruktoru pohledu, kterým byl `QTreeView` - základní stromový pohled Qt. Brzy se ale ukázalo, že na takovém modelu se samotná aplikace bude velmi špatně testovat a vyvíjet, protože je potřeba testovat i změny ve stavech kontaktů a především pak ovládání samotného online stavu aktivních účtů (o tom více v části Seznam konatků). Samozřejmě by se dal implementovat kompletní falešný model simulující skutečný model kontaktů, vývoj investovaný do takovéhoto prototypu by ale byl daleko lépe investovaný do vývoje skutečného modelu.

3.5.1.2 AccountsModel Ukázalo se, že víceméně funkční základní model kontaktů založený na `QAbstractItemModel` už existuje v knihovně Telepathy-Qt-Yell, která slouží jako knihovna pro experimentální vývoj nových funkcí, které se po dokončení a stabilizaci přesunují do Telepathy-Qt. Model, nazvaný `AccountsModel`, svou architekturou i licencí vyhovoval potřebám projektu a tak nemělo smysl vytvářet stejnou věc znovu. Adaptoval jsem jej tedy pro použití v KDE Telepathy a začal nad ním vyvíjet vlastní seznam kontaktů.

Model seskupoval kontakty podle jejich účtů a měl v podstatě stejnou architekturu jako můj předešlý falešný model. Prvky stromu tvořil objekt `TreeNode`. Každý `TreeNode` měl ukazatel na objekt nad sebou („rodič“) a všechny svoje podobjekty (tedy čí byl daný `TreeNode` rodič). Model samotný pak měl pouze ukazatel na kořenový `TreeNode`. Třída `TreeNode` rozšiřovala třídu `QObject` a implementovala `Q_OBJECT` makro. To znamenalo, že `TreeNode` objekty mohly využívat mechanizmu signálů a slotů. Právě pomocí signálu implementoval `TreeNode` řetězovou propagaci změn až k samotnému modelu, který na změny reagoval. Signály byly tři - `changed(TreeNode*)`, `childrenAdded(TreeNode *parent, QList<TreeNode*> children)` a `childrenRemoved(TreeNode *parent, int from, int to)`. První signál sig-

nalizuje změnu samotných dat objektu, který byl předán jako parametr. Model nejprve převedl obdržený objekt `TreeNode` na index a ten signalizoval pohledu, který změněná data překreslil. Signál `childrenAdded()` sloužil k oznamování modelu, že se mají vložit nová data, která se předávala jako parametry - prvním byl rodiče a pak seznam objektů, které se k němu mají připojit. Implementace `TreeNode` tedy pouze připravila vložení, samotné vkládání ale prováděl model. To proto, že bylo potřeba pomocí rozhraní modelu signalizovat pohledu, že se budou vkládat nová data. Signál `childrenRemoved()` pak plnil obrácenou funkci - signalizoval modelu, které potomky daného rodiče se mají odstranit. Tentokrát se předával pouze rozsah, nikoliv samotné objekty ke smazání. Bylo nutné znát rozsah, protože model potřeboval oznámit pohledu, které položky z něj budou odstraněny.

Pro získání dat z modelu slouží metoda `QAbstractItemModel::data(QModelIndex index, int role)`, kterou volá pohled (ale nejen ten). Volající objekt jako parametry předá index, pro který data požaduje, a roli, tedy jaká data chce (role jsou reprezentovány jako enum, tedy výčet intů). V našem modelu metoda `data()` namapovala předaný index na konkrétní objekt stromové struktury a zavolala metodu `data()` na něm. Výchozí implementace `TreeNode` vracela vždy prázdnou hodnotu.

Samotná třída `TreeNode` byla spíše abstraktní třídou implementující pouze stromovou strukturu, pro konkrétní objekty sloužily rozšiřující třídy `ContactModelItem` a `AccountsModelItem`. Tyto třídy pak sloužily jako přímé rozhraní pro objekty `Tp::Contact`, resp. `Tp::Account`.

Každý `ContactModelItem` uchovával jako třídní proměnnou právě jeden `Tp::ContactPtr` a byl připojen ke všem jeho signálům. Každá změna dat, kterou `Tp::Contact` signalizoval, byla díky řetězové propagaci změn doručena až do pohledu, který na signál reagoval novým vykreslením položky v pohledu. Změnil-li kontakt např. svoji fotku, byla změna okamžitě vidět. `ContactModelItem` reimplementuje metodu `data()`, ve které jednotlivé role mapuje na vlastnosti objektu kontaktu.

`AccountsModelItem` je pak v podstatě to samé, co `ContactModelItem`, ale pro účty, ke kterým kontakty patří. `AccountsModelItem` uchovává `Tp::AccountPtr` jako třídní proměnnou a je také připojen ke všem jeho signálům. Taktéž jako u `ContactModelItem` je změna propagována přímo až do modelu, kde se přemapuje objekt vysílající signál na index a ten se pošle pohledu. I `AccountsModelItem` implementuje metodu `data()`, ve které mapuje role na vlastnosti účtu. Každý `AccountsModelItem` je pak zodpovědný za vytvoření objektů třídy `ContactModelItem` pro všechny kontakty, které účet obsahuje.

Vztah mezi účty a kontakty v Telepathy je nepřímý a v podstatě pouze jednostranný - účet vytvoří spojení, jehož součástí je objekt Contact Manager, který spravuje seznam všech kontaktů daného účtu. Contact Manager existuje pouze po dobu života spojení. Zanikne-li z jakýchkoliv důvodů spojení (např. výpadek sítě), zanikne i Contact Manager a všechny jeho kontakty jsou zneplatněny. Je tedy zřejmé, že kontakty v Telepathy nejsou vázány na účet, ale na jeho spojení. Z toho plyne několik důsledků - např. uživatel nemá k dispozici seznam kontaktů, pokud není daný účet připojen. Navíc z objektu kontaktu

nedokážeme zjistit, ke kterému účtu patří. Můžeme z něj získat objekt Contact Manageru, který jej spravuje, a z něj pak i spojení, na kterém Contact Manager existuje. Z objektu spojení už ale nelze získat účet, ke kterému spojení patří. Naopak ze strany účtu objekt konkrétního kontaktu získat umíme, neboť umíme získat objekt spojení, z něho objekt Contact Manageru a z něj pak konkrétní kontakt.

Tuto situaci částečně řešil `AccountsModel`, resp. `AccountsModelItem`. Ten z Contact Manageru získal všechny kontakty a pro každý z nich vytvořil `ContactModelItem`. Poté vyslal signál `childrenAdded(this, QList<TreeNodes*>)`, což jej efektivně učinilo rodičem všech objektů `ContactModelItem`. Díky této hierarchii bylo možné přímo zjistit, ke kterému účtu daný kontakt náleží. `AccountsModelItem` musel samozřejmě také implementovat správu spojení. Pokud spojení zaniklo, musely se odstranit všechny `ContactModelItem` objekty, kterým byl `AccountsModelItem` rodičem (samotné objekty `Tp::Contact`, které `ContactModelItem` držel jako třídní proměnné, byly zaniklým spojením zneplatněny). Pokud bylo naopak navázáno nové spojení, `AccountsModelItem` znovu vytvořil `ContactModelItem` pro každý kontakt. Protože `ContactModelItem` byly malé a jednoduché objekty, byla operace velmi rychlá.

`AccountsModelItem` oproti třídě `ContactModelItem` dále implementoval i metody pro přímou manipulaci účtu jako např. změnu jeho statusu.

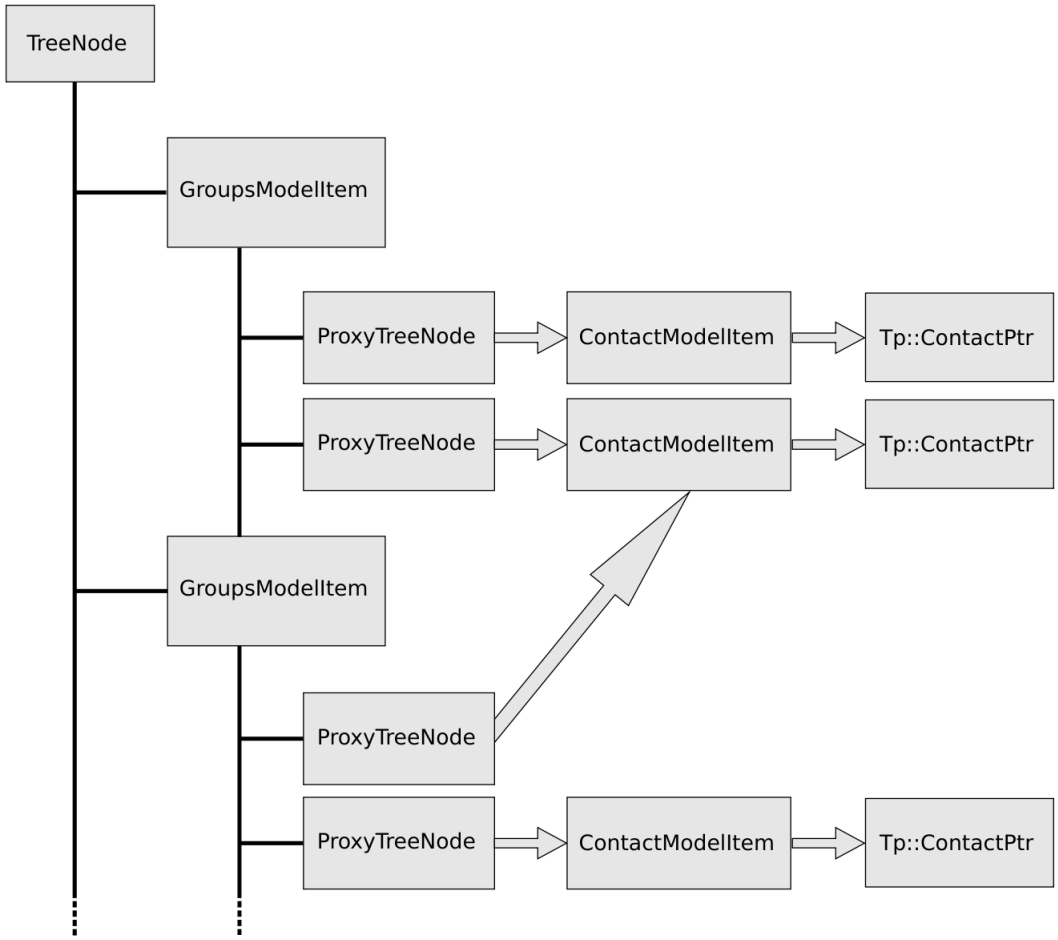
`AccountsModel` při svém vzniku vytvořil obyčejný `TreeNode` objekt, který sloužil jako kořenový prvek celé stromové struktury. Skrze tento kořenový `TreeNode` objekt se modelu předávaly veškeré signály z celého stromu kontaktů. Ke své funkci potřeboval model připravený Account Manager, ze kterého získal seznam všech uživatelových účtů, a poté pro každý účet vytvořil nový objekt `AccountsModelItem`, který přidal jako potomek kořenového objektu `TreeNode`. V konstruktoru třídy `AccountsModelItem` se pak vytvářely instance třídy `ContactModelItem`, a tak vznikla celá struktura modelu.

3.5.1.3 GroupsModel

`AccountsModel` seskupoval kontakty podle účtu, ke kterému patřily. Kontakty ale mají v rámci účtu i vlastní skupiny, do kterých patří. Dalším úkolem tedy bylo vyvinout model, který by seskupoval kontakty podle jejich skupin, nezávisle na účtu, ke kterému patří.

`GroupsModel` byl nádstavba nad `AccountsModelem`, který potřeboval ke svému fungování. Dědil ze základní třídy `QAbstractItemModel` a stejně jako `AccountsModel` měl stromovou strukturu složenou z objektů třídy `TreeNode`. Kontakty představoval objekt `ProxyTreeNode` (který samozřejmě dědil z třídy `TreeNode`), jenž sloužil jako proxy objekt pro `ContactModelItem` z `AccountsModelu`. Jednotlivé objekty `ProxyTreeNode` poté byly připojeny jako potomci pod příslušný `GroupsModelItem`, což byla třída reprezentující skupinu.

Konstruktoru modelu byl předán `AccountsModel`, který se celý procházel a pro každý kontakt z něj byl vytvořen nový objekt `ProxyTreeNode` a ten přidán do nového modelu. `ProxyTreeNode` pak měl jako třídní proměnnou ukazatel na příslušný `ContactModelItem`. Díky použití proxy objektu bylo možné umístit jeden kontakt i do více skupin - každá skupina by jako potomka měla vlastní `ProxyTreeNode`, přičemž všechny tyto dané objekty by měly ukazatel na stejný `ContactModelItem` objekt a re-



Obrázek 3: Hierarchie datové struktury modelu GroupsModel

prezentovaly tak stejný kontakt. `ProxyTreeNode` byl připojen přímo k signálům objektu `Tp::Contact`, který držel `ContactModelItem`. Dále `ProxyTreeNode` naslouchal signálům `destroyed()` a `changed()` objektu `ContactModelItem`. Signál `destroyed()` signalizuje, že objekt byl smazán. Pokud byl `ContactModelItem` smazán, musel být smazán i `ProxyTreeNode`, neboť by držel ukazatel na neplatný objekt a způsobil pády aplikace. Signál `changed()` byl stejně jako v `AccountsModelu` skrz rodiče daného `ProxyTreeNode` propagován dále až do modelu.

`Tp::Contact` implementuje metodu `groups()`, která vrací všechny skupiny, do kterých daný kontakt náleží. Při přidávání kontaktů do modelu se tak nejprve prohledal již existující strom do první úrovně, zda-li neobsahuje již vytvořené objekty pro skupiny (`GroupsModelItem`), do kterých kontakt patřil. Pokud nebyl nalezen existující objekt `GroupsModelItem` pro danou skupinu, byl vytvořen nový a kontakty (resp. `ProxyTreeNode` objekty) byly přidány jako jeho potomci. Pokud kontakt nebyl přiřazen do žádných skupin (`Tp::Contact::groups()` vracelo prázdný seznam), byl zařazen

do speciální skupiny s názvem „Nezařazené kontakty“.

Veškeré změny struktury prováděl opět sám model, který pomocí speciálních rozhraní musel nejprve uvědomit pohled, že se budou vkládat nebo odstraňovat data.

3.5.1.4 AccountsFilterModel Poté, co byl základní model hotov, bylo potřeba do aplikace pro seznam kontaktů přidat řazení kontaktů, funkce vyhledávání v kontaktech a skrývání nepřipojených kontaktů. Pro vyhledávací a filtrovací operace nad modely nabízí Qt třídu `QSortFilterProxyModel`. Tato třída se vkládá mezi model a pohled a v podstatě mapuje indexy zdrojového modelu na nové indexy, přičemž každý index zdrojového modelu je zkontrolován proti nastavenému filtru. Pokud index filtru nevyhovuje, není předán pohledu a ten tak data (v našem případě kontakt) ležící na pozici indexu nezobrazí. Potřebujeme-li složitější filtrování či vlastní mechanizmus řazení, je třeba třídu `QSortFilterProxyModel` rozšířit a reimplementovat metody `filterAcceptsRow(int row, QModelIndex index)` a `lessThan(QModelIndex left, QModelIndex right)`.

`AccountsFilterModel` tedy rozšiřuje třídu `QSortFilterProxyModel` a poskytuje metodu pro zapnutí/vypnutí filtrace nepřipojených kontaktů (připojenou k signálu tlačítka v aplikaci) a také pro jednoduché vyhledávání v kontaktech. Dále poskytuje možnost řadit buďto podle jména kontaktů nebo podle jejich stavu.

Filtraci zajišťuje metoda `filterAcceptsRow()`, které je předán index rodiče ze zdrojového modelu a číslo řádku jeho potomka, přičemž se zpracovává vždy pouze to, co zobrazuje pohled. Jelikož se jedná o stromovou strukturu, budou jednotlivé položky modelu zpracovány filtrem až při rozbalení jejich nadřazené položky v pohledu. Představíme-li si strukturu našeho `AccountsModelu`, v první fázi bude předán index s pozicí (0,0), což odpovídá kořenovému objektu `TreeNode`. Jeho řádky jsou pak jednotlivé účty (resp. objekty `AccountsModelItem`). Po rozbalení položky účtu v pohledu se metodě `filterAcceptsRow()` předá index s pozicí daného účtu v modelu a řádky jsou pak jeho jednotlivé kontakty.

Když máme index rodiče a pozici potomka, můžeme sestavit nový index, který ukazuje přímo na samotná data potomka. V reimplementované metodě `filterAcceptsRow()` se tedy nejprve sestavil nový index a poté se zkontrolovalo, o jaký druh objektu se jedná - kontakt, účet či skupina (resp. `ContactModelItem`, `AccountsModelItem` či `GroupsModelItem`). Podle toho se lišilo vlastní filtrování. Při filtrování kontaktů se nejprve zkontrolovalo, zda-li `AccountsFilterModel` nemá nastaven vyhledávací řetězec, pocházející typicky z vyhledávacího pole aplikace. Pokud aktuální kontakt vyhledávaný řetězec neobsahoval, byl vyfiltrován pryč. Jestliže měl `AccountsFilterModel` zapnuto filtrování nepřipojených kontaktů, byl dále z každého indexu získán aktuální stav kontaktu a pokud byl stav „nepřipojen“, kontakt byl také vyfiltrován pryč.

Z účtů se vyfiltrovaly ty účty, které byly v Account Manageru označeny jako zakázané. Ze skupin se vyfiltrovaly pryč ty skupiny, které v sobě neměly žádné kontakty. To záleželo také na tom, zda-li bylo zapnuto zobrazování nepřipojených kontaktů - pokud v jedné

skupině byly všechny kontakty ve stavu offline a zobrazování nepřipojených kontaktů bylo vypnuto, skupina byla prázdná a tudíž filtrem nepropuštěna.

Pro řazení podle stavu kontaktů bylo potřeba reimplementovat metodu `bool lessThan(QModelIndex left, QModelIndex right)`. Metodě jsou předány dva indexy, které se porovnávají a implementace musí vrátit `true`, pokud je první index menší než druhý, `false` v opačném případě. `AccountsFilterModel` nejprve zkontroloval, podle čeho se má řadit. V případě stavu kontaktů se z indexů získaly oba stavy a ty se porovnávaly podle předdefinovaných priorit, přičemž nejvíce online stav měl prioritu 0, stav offline prioritu 6. Pokud byly oba stavy stejné, porovnaly se jména kontaktů pomocí funkce
`QString::localAwareCompare(QString left, QString right)`, které se předaly obě jména a stejně jako metoda `lessThan()`, vrátila `true`, pokud byl první řetězec menší než druhý, jinak vrátila `false`. V případě, že bylo nastaveno řazení podle jmen, jednoduše se porovnaly obě jména opět pomocí `QString::localAwareCompare()`.

Ve verzi 0.4 se pak objevuje komplexnější verze tohoto filtru, která umožňuje komplexní filtrování kontaktů podle libovolného stavu (např. pokud chceme pouze kontakty ve stavu „zaneprázdněn“) či některých vlastností jako např. schopnosti kontaktu (video hovory, přenosy souborů atp.) nebo je-li kontakt blokován. K filtrování účtů přibyla možnost odfiltrovat nepřipojené účty. Konfigurace proxy modelu se prováděla pomocí příznaků (flags). Qt framework práci s příznaky usnadňuje automatickým vytvořením nového typu a přidáním typové kontroly při kompilaci. To zajišťuje, že lze uložit kombinace pouze jednoho výčtu, při použití hodnot jiného výčtu kompilace selže. Pro použití příznaků stačí definovat výčet (enum) s hodnotami, kde každá hodnota představuje jeden bit v bajtu (či více bajtech) paměti, a přidat makro `Q_DECLARE_FLAGS`.

V C++ proměnná typu bool standardně zabírá celý bajt paměti i přesto, že by stačil jediný bit. Díky příznakům lze efektivně uložit 8 booleovských hodnot v jednom bajtu paměti. Další z důvodů, proč jsme použili příznaky, byla jednodušší konfigurace v klientském kódu. Pokud by byla každá volba uložena jako samostatná proměnná typu bool, musela by pro každou volbu existovat metoda, která danou proměnnou nastaví. To znamená 34 přídavných metod pro nastavení a pravděpodobně dalších 34 metod pro zjištění daného nastavení. Tedy celkem 68 metod pouze pro nastavení v API `AccountsFilterModelu`. Konfigurace pomocí příznaků tak je nejen jednodušší, ale i přehlednější. Příznaky jsou uloženy jako OR kombinace hodnot. Kontrola příznaků se pak provádí pomocí bitové operace AND.

3.5.1.5 ContactListModel Pro vydání verze 0.6 jsou pak modely `AccountsModel` a `GroupsModel` od základu přepracovány. Základem se stává `ContactListModel`, který je vystavěn na třídě `KTp::GlobalContactManager`, což je jednoduché rozhraní pro získání všech kontaktů z aktuálně připojených účtů. Navíc poskytuje způsob, kterým lze získat účet, ke kterému kontakt patří, na základě spojení. Jak již bylo popsáno v kapitole 3.5.1.2, každý kontakt (reprezentován objektem `Tp::Contact`) drží ukazatel na svůj Contact Manager (reprezentován objektem `Tp::ContactManager`), který drží ukazatel

na spojení, na kterém existuje. Náš `GlobalContactManager` tedy jednoduše z kontaktu získá spojení a podle toho najde příslušný účet.

Samotný model pak udržuje seznam kontaktů v podobě objektů `KTp::Contact`. Tato třída rozšiřuje třídu `Tp::Contact` a přidává pár velmi užitečných funkcí jako např. kontrolu schopností, které kontakt má. Dříve totiž kontrolu schopnosti zajišťoval každý klientský kód sám a metody se mnohdy různily. Proto jsme se rozhodli napsat metodu jednu a spolehlivou a tu dát jako vlastnost samotnému kontaktu. Později přibyla i metoda pro získání profilového obrázku kontaktu i když je kontakt odpojen (Telepathy standardně obrázky pro offline kontakty nevrací, Telepathy-Qt je však ukládá do lokální cache, ze které je možné je získat). Navíc metoda vrací profilové obrázky v odstínech šedi, je-li kontakt odpojen. `KTp::Contact` je součástí ktp-common-internals od verze 0.6.

Model rozšiřuje třídu `QAbstractListModel`. Už tedy netvoří strom, ale obyčejný list. To je výhodné i pro situace, kdy chceme zobrazit pouze kontakty bez jakéhokoliv rozdělení na skupiny či podle účtů. Stromová struktura pro zobrazení skupin a účtů je realizována až na úrovni proxy modelu, který funguje nad tímto modelem. Jednotlivé role modelu byly také pročištěny a ponechány pouze ty, kterých je skutečně potřeba. Navíc byly přesunuty do samostatného hlavičkového souboru, který je možné vkládat z různých míst a používat tak na všech místech stejné role bez nutnosti vkládání hlavičkového souboru celého modelu.

3.5.1.6 AbstractGroupingProxyModel Tento model je založen na stejném principu jako `GroupsModel`, jeho fungování je ale sofistikovanější a obecnější. To znamená, že tento model lze použít s jakýmkoliv modelem, není již vázán pouze na `AccountsModel`. Model rozšiřuje třídu `QStandardItemModel` a není tak přímo proxy modelem (nedědí ze tříd pro proxy modely), svým principem (neposkytuje žádná data, pouze přetváří jiný model) ale ze sebe efektivně dělá proxy model. `QStandardItemModel` implementuje rozhraní `QAbstractItemModelu` a poskytuje jednoduchý, rychlý a pohodlný způsob, jak vytvořit model. Postrádá sice flexibilitu (a částečně i výkon) vlastních implementací `QAbstractItemModelu`, ty však v tomto případě nebyly potřeba a zjednodušené rozhraní naopak přišlo vhod.

`AbstractGroupingProxyModel` pracuje se dvěma objekty - `ProxyNode` a `GroupNode`. Oba rozšiřují třídu `QStandardItem`, která je datovým objektem `QStandardItemModelu`. `QStandardItem` poskytuje jednoduché rozhraní pro vytváření libovolných struktur - stromů, tabulek, obyčejných seznamů, ale i kombinaci všech tří dohromady. Model samotný deklaruje dvě čistě virtuální metody - `QSet<QString> groupsForIndex(QModelIndex)` a `QVariant dataForGroup(QString group, int role)`. Metodou `groupsForIndex()` lze definovat, podle čeho budou jednotlivé položky modelu seskupeny. Metoda z předaného indexu získá potřebná data a ty vrátí jako množinu `QSet` (oproti seznamu typu `QList` nemá množina nikdy duplicitní položky). Metoda `dataForGroup()` pak vrací data podle role pro konkrétní objekt skupiny.

`ProxyNode` je jednoduchá třída, která v modelu reprezentuje prvek zdrojového modelu, jehož index ukládá jako třídní proměnnou. Implementuje pouze tři metody

- `QVariant data(int role)`, `changed()` a `QString group()`. Metoda `data()` slouží k získání dat prvku zdrojového modelu, k tomu se přímo využívá uložený index, kterému se předá role. Metoda `changed()` pouze zpřístupňuje `protected` metodu `emitDataChanged()` základní třídy a slouží k signalizování modelu, že daný objekt byl změněn. `QStandardItem` k propagaci změn nepoužívá signálů a slotů, ale jednoduchý mechanismus, při kterém se modelu přímo předá změněný objekt a model si pro objekt najde index, jehož změnu pak signalizuje pohledu. Objekty modelu tak nemusí rozšiřovat `QObject` a implementovat `Q_OBJECT` makro, díky čemuž jsou tyto objekty menší a rychlejší. Třetí metoda, `group()`, vrací název skupiny, do které objekt patří, popř. prázdný řetězec, pokud nemá nadřazenou skupinu.

Druhý objekt modelu, `GroupNode`, reprezentuje objekt, pod kterým jsou seskupeny prvky mající určitý stejný parametr. Tento parametr je definován rozšiřujícími třídami našeho modelu. `GroupNode` je podobně jednoduchá třída jako `ProxyNode` - rozšiřuje `QStandardItem`, implementuje metody `changed()` a `QVariant data(int role)` a navíc ukládá parametr „forced“, který určuje, že skupina nebude z modelu odstraněna, když už neobsahuje žádné prvky.

Samotný `AbstractGroupingProxyModel` pak prochází zdrojový model, kterým je v našem případě `ContactsListModel` a pro každý prvek zdrojového modelu nejprve pomocí metody `groupsForIndex()` získá všechny skupiny, do kterých aktuálně zpracovávaný prvek patří, a poté do každé skupiny přidá jeden objekt `ProxyNode`, přičemž zároveň vytváří objekt dané skupiny, pokud ještě neexistuje. `AbstractGroupingProxyModel` je samozřejmě připojen k signálům zdrojového modelu a na změny modelu patřičně reaguje - např. když kontakt změní skupinu, je potřeba nalézt ty objekty `ProxyNode`, které odpovídají danému kontaktu ve staré skupině a buďto je přesunout do nové skupiny, nebo pouze je z modelu odstranit, pokud už v nové skupině existují. Kontakty, které nepatří do žádné skupiny, jsou automaticky umístěny do skupiny „Nezařazeno“.

V KDE Telepathy používáme dvě rozšiřující třídy pro `AbstractGroupingProxyModel` - `AccountsTreeProxyModel` a `GroupsTreeProxyModel`. Obě třídy implementují virtuální metody `groupsForIndex()` a `dataForGroup()` (viz výše). Tyto třídy zajišťují seskupování podle účtů, ke kterým kontakty patří, resp. podle skupin. `AccountsTreeProxyModel` navíc implementuje velmi jednoduché rozhraní pro sledování změn účtů - pokud je změněna ikona účtu nebo jeho zobrazované jméno, je signalizována změna seskupujícího `GroupNode` objektu reprezentujícího daný účet, čímž se zajistí promítnutí změn v pohledu.

Od verze 0.3 jsou modely odděleny do samostatné knihovny v rámci ktp-common-internals, od verze 0.5 v rámci ktp-common-internals vzniká i další samostatná knihovna sdružující grafické ovládací prvky. S verzí 0.6 pak přibyla nová knihovna pro podporu uživatelských rozhraní v jazyce QML.

Ve verzi 0.4 se do ktp-common-internals přesouvají také všechny ikony projektu, které je potřeba nainstalovat. Verze 0.5 pak přináší i nový speciální nástroj nazvaný ktp-debugger, kterým je možné jednoduše získat logy jednotlivých Connection Managerů,

které pomáhají při řešení chyb a problémů hlášených uživateli.

3.5.1.7 KTp::Presence Za zmínku stojí také třída `KTp::Presence`, která rozšiřuje třídu `Tp::Presence`. Tyto třídy slouží pro reprezentaci statusu, ať už jednotlivých kontaktů nebo účtů. Třída byla vytvořena především pro usnadnění reprezentace statusu v uživatelských rozhraních, kde každý kód zobrazující status musel sám přiřadit ikony odpovídající statusu, popř. také zobrazované názvy statusů. Obzvlášť názvy byly problém, protože každá třída, která tuto část kódu duplikovala, znamenala také duplikaci řetězců pro překlady a překladatelé tak museli stejný řetězec překládat několikrát a nebylo možné zaručit, že všechny verze budou přeloženy stejně. `KTp::Presence` tak vlastně přidává pouze metody pro získání ikony a zobrazitelného, přeloženého názvu konkrétního stavu, a navíc implementuje operátor ¡, takže je možné statusy porovnávat. Ve starších verzích knihovny se na všech místech v klientské aplikaci musel objekt `KTp::Presence` tvořit explicitně z původní `Tp::Presence`, kterou vrací `Tp::Contact`; kód pro získání ikony tedy vypadal např. takto:

```
Tp::ContactPtr contact = index.data(KTp::ContactRole).value<Tp::ContactPtr>();
KTp::Presence presence = KTp::Presence(contact->presence());
KIcon presenceIcon = presence.icon();
```

S nasazením třídy `KTp::Contact` pro kontakty však toto odpadá a `KTp::Presence` vytváří a vrací už sám objekt `KTp::Contact`; kód uvedený výše lze tedy zjednodušit na toto:

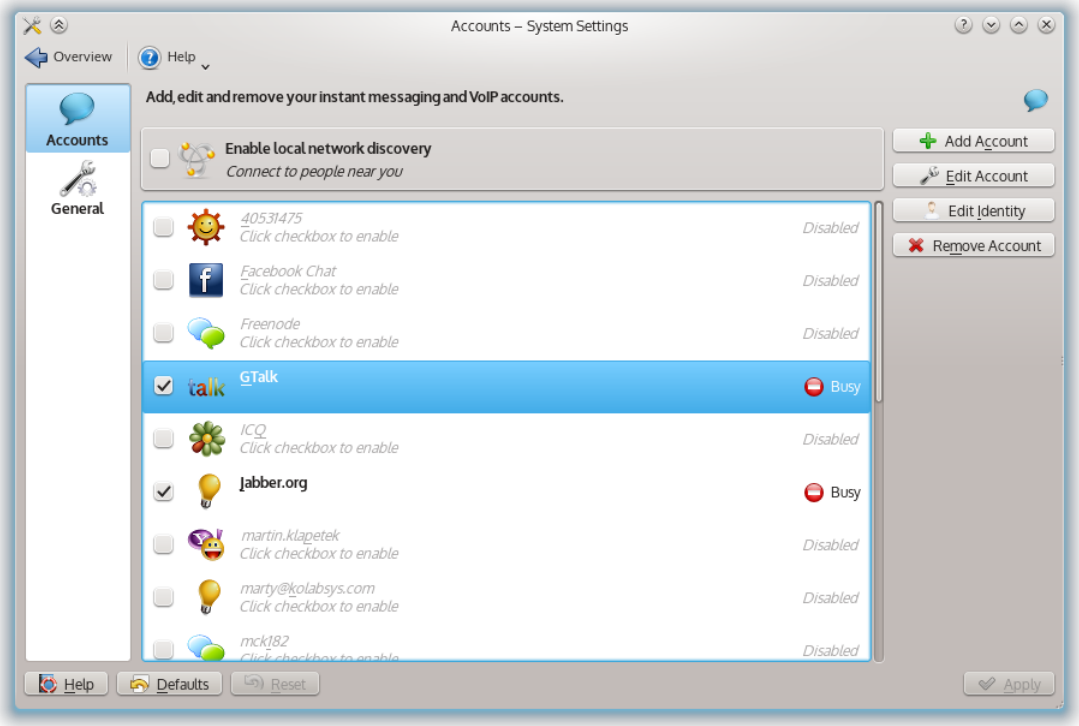
```
KTp::ContactPtr contact = index.data(KTp::ContactRole).value<KTp::ContactPtr>();
KIcon presenceIcon = contact->presence().icon();
```

Podle nástroje CLOC má ktp-common-internals v aktuálně poslední vydané verzi (0.6.1) 9 460 řádků čistého kódu (bez komentářů a „bílých míst“).

3.5.2 Správa účtů

Komponenta slouží pro přidání, nastavení a správu účtů IM sítí. Je implementována jako KConfig Modul (KCM), díky čemuž je možné ji zobrazit jako položku v centrálním nastavení KDE. K tomu je potřeba čtyř věcí - hlavní třída modulu musí dědit z třídy `KCModule`, modul musí být zkompilován jako sdílená knihovna, modul musí být exportován (lze použít `KPluginFactory`) a modul musí instalovat .desktop soubor se svými informacemi, aby systém nastavení o modulu věděl.

Správa účtu poskytuje uživateli velmi jednoduché rozhraní pro přidávání známých účtů jako např GTalk, Facebook Chat nebo Jabber. Přidání nového účtu je realizováno pomocí průvodce - v prvním kroku si uživatel vybere, ke které síti se chce připojit, ve druhém zadá přihlašovací jméno a heslo s možností zobrazení pokročilejší konfigurace, potřebuje-li ji uživatel. Po potvrzení průvodce se ve výchozím stavu účet automaticky připojí. Protože se ale ozývali uživatelé, že se jim toto chování nelíbí z důvodu bezpečnosti („Co když zrovna nechci dát mým kontaktům vědět, že jsem u počítače“ či „nemám možnost připojování nijak zastavit“ atp.), bylo do druhého kroku přidáno zaškrtávací políčko „Připojit



Obrázek 4: Správa účtů

po dokončení “, které je ve výchozím stavu zaškrtnuto. Uživatelé s jakýmikoli obavami tak můžou volbu jednoduše zrušit.

Každý účet v Telepathy může být aktivní („enabled“) nebo neaktivní („disabled“). Neaktivní účty se nikdy nepřipojí (např. při nastavení automatického připojování) a je nutné je nejprve aktivovat před navázáním jakéhokoliv spojení. Je-li účet přepnut do neaktivního stavu, je samozřejmě okamžitě odpojen, je-li přepnut do stavu aktivního, Správa účtů KDE Telepathy jej okamžitě automaticky připojí (jaký smysl by pak aktivace účtu měla, kdyby uživatel nechtěl účet připojit?). Aktuální stav každého účtu je v modulu zobrazen - popiskem, je-li účet neaktivní, v opačném případě ikonou a názvem aktuálního statusu. Vyskytne-li se během připojení jakákoliv chyba, je pod názvem účtu zobrazen uživatelsky přívětivý název chyby (spolu se zobrazením systémové notifikace).

U připojených účtů je možné změnit jméno a avatar zobrazované kontaktům. Ne všechny sítě toto ale podporují - např. Facebook Chat, což je velmi omezená implementace protokolu XMPP, má tyto údaje (a vlastně všechny údaje) pro XMPP klienty pouze ke čtení, uživatel je tak nemůže změnit.

Telepathy umožňuje také nastavení vlastního jména a ikony účtu, které pak uživatelské rozhraní zobrazuje. Místo zobrazování složitého uživatelského jména si tak uživatel může nastavit jednoduchý popisek. To samé platí pro ikonu, která je při identifikaci

daného účtu ve větším počtu účtů hlavním rozpoznávacím prvkem. Sdílená knihovna ktp-common-internals instaluje ikony pro většinu známých sítí a tyto ikony jsou použity jako výchozí, aby i ve výchozím stavu bylo velmi snadné poznat daný účet (každý si pamatuje, že Facebook je bílé „f“ v modrém poli, ICQ je zelená květinka atd.).

3.5.3 Aplikace schvalující příchozí kanály

Tato komponenta je v architektuře Telepathy schvalujícím klientem. Jejím úkolem není nic jiného, než informovat uživatele o nových příchozích kanálech a dát mu možnost na ně reagovat. Komponenta vytvoří jednak systémovou notifikaci zobrazující text příchozí zprávy (popř. jméno volajícího nebo název souboru, který nám někdo posílá) a jednak vytvoří „Status Notifier Item“ (SNI), což je obdoba starších klasických ikon v „systrayi“. Specifikace SNI byla vytvořena jako doplněk specifikace notifikací a náhrada za starou specifikaci systraye. Hlavní myšlenkou je tedy zobrazit notifikaci a přidat do systémové části panelu ikonu nové zprávy. To má hned několik výhod - pokud si uživatel příchozí notifikace nevšimne (např. je pryč od počítače) nebo nechce reagovat okamžitě, blikající ikona v systrayi dále oznamuje novou zprávu a dává uživateli možnost reagovat kdy on sám chce/může. Díky architektuře Telepathy „komunikace jako služba“ nemusí existovat žádná ikona v systrayi, která ukrývá celou aplikaci, jak tomu je u jiných klientů. Ti obvykle zobrazují novou zprávu právě na oné ikoně v systrayi. Vizí za SNI byla lepší integrace ikon v systrayi v jednotlivých prostředích - oznámení o příchozí zprávě tak může vypadat úplně jinak v KDE oproti GNOME a přitom stále poskytovat úplně stejnou funkcionalitu. SNI se bohužel i přes dlouhou diskuzi mimo KDE nikdy nerozšířilo. Výhodou SNI oproti jedné ikoně v systrayi „na všechno“ je také možnost vytvářet více SNI najednou - schvalující komponenta vytváří SNI zvlášť pro textové zprávy a příchozí soubory.

Komponenta je realizována jako modul pro KDED, což je proces běžící na pozadí, který sdružuje více menších, převážně jednoúčelových služeb do jednoho procesu. Původní záměr byl mít tuto komponentu neustále aktivní a naslouchající příchozím kanálům; KDED, který běží po celou dobu od přihlášení do odhlášení, je proto ideální místo. Později byla tato schvalující komponenta upravena, aby mohla být automaticky spuštěna až v případě potřeby, i přesto ale zůstala realizována jako KDED modul, jednoduše nebyl důvod to měnit.

Přijme-li uživatel příchozí zprávu, ať už kliknutím na notifikaci či SNI, schvalovací komponenta prohledá seznam možných obslužných klientů a porovná jej s předdefinovaným seznamem preferovaných obslužných klientů. Pokud preferovaní klienti nejsou v seznamu dostupných klientů nalezeni, jsou ze seznamu odstraněni. Nakonec se oba seznamy sjednotí s tím, že preferovaní klienti jsou v seznamu jako první. Pak už se jen kanál předá klientovi k obsluze. Pokud předání kanálu selže, schvalovací komponenta zkusí dalšího klienta ze seznamu. Vyčerpá-li seznam dostupných klientů, svou činnost ukončí a Channel Dispatcher kanál uzavře.

3.5.4 Aplikace poskytující autentizační údaje ze systémového úložiště KWallet

Pro autentizaci uživatelů vytváří Telepathy kanálové rozhraní, lze s ní nakládat úplně stejně jako s jinými kanály. Tato komponenta obsluhuje právě autentizační kanály, které Telepathy vytváří. Aktuálně je schopna obsloužit kanály typu SASL, TLS, OAUTH a autentizaci pro uzamčené skupinové chaty. Z pohledu klienta se jedná o standardní věc - malá aplikace, kterou Telepathy automaticky spustí v případě potřeby a předá kanál k obsluze.

Aplikace z prostředí KDE Plasma standardně ukládají hesla do sdíleného úložiště nazvaného KWallet. KDE Telepathy se v duchu integrace nechová jinak. Při předání SASL kanálu tedy komponenta nejprve otevře úschovnu KWallet, pokud operace skončí úspěšně, přečtou se vlastnosti kanálu a podle druhu ověřovacího mechanismu se spustí příslušná operace, která přečte autentizační informace z KWallet a předá je SASL rozhraní. Pokud autentizace selže, je z vlastností kanálu přečtena hodnota „CanTryAgain“, tedy můžeme-li se pokusit znovu autentizovat. Pokud ano, zobrazí se uživateli výzva k zadání nového hesla (protože s heslem uloženým se autentizovat nepodařilo) a proces se opakuje. Pokud kanál předá `CanTryAgain` jako `false`, komponenta informaci uloží do KWallet, kanál uzavře a uměle vyvolá nové připojení, tím se vytvoří nový SASL kanál a nový pokus o autentizaci. Pokud KWallet obsahuje informaci, že se poslední autentizace nezdařila, je uživatel rovnou vyzván k zadání nového hesla. Uživatel tak odmítnutí dalšího pokusu o autentizaci vůbec nepozná. Kromě autentizace heslem umí komponenta provést autentizaci pomocí OAuth2 mechanismu. To je aktuálně použito pouze pro síť MSN, pracuje se ale i na podpoře pro Google Talk. Výhodou OAuth2 autentizace je, že uživatel aplikaci nikdy neposkytuje své heslo, aplikace pouze otevře webové rozhraní dané služby a uživatel se svým heslem autentizuje tam. Obratem získá autentizační řetězec (OAuth Access Token), který je pak aplikací použit k autentizaci namísto hesla.

Při obdržení TLS kanálu, který obsahuje certifikační řetězec pro zabezpečenou komunikaci, je potřeba ověřit obdržený certifikát oproti systémovým certifikátům. KDELibs k tomu mají potřebné nástroje, ale neinstalují pro ně hlavičkové soubory, což je dělá efektivně nepoužitelnými mimo KDELibs. Tento problém byl diskutován s vývojáři KDELibs, ti však trvali na tom, že potřebné třídy modulu KSSL jsou určeny pouze pro interní použití v KDELibs. Proto byly potřebné hlavičkové soubory i přes všechna rizika zkopírovány a použity lokálně. Díky tomu může komponenta certifikáty ověřit a v případě problémů informovat uživatele a umožnit mu šifrované spojení i přes varování o problémech s certifikátem.

Posledním druhem kanálů, které komponenta obsluhuje, jsou heslem chráněné skupinové chaty („conference rooms“). Mechanismus je velmi podobný obsluze SASL kanálů. Nejprve se otevře úložiště hesel KWallet, pokud má uložené heslo, tak se použije, pokud nemá, nebo pokud je heslo nesprávné, je uživatel vyzván k zadání nového hesla. Pokud se podaří uživatele novým heslem ověřit, je heslo uloženo pro příští použití.

3.5.5 Seznam kontaktů

Tato aplikace představuje klasický seznam kontaktů, jak jej známe z jiných IM klientů. Aplikace umožňuje základní správu kontaktů jako přidávání, mazání, blokování či přesun mezi skupinami, a také umožňuje nastavení aktuálního IM stavu. Tato aplikace byl poslední chybějící článek k první použitelné verzi KDE Telepathy. Po mém připojení do týmu jsem ji proto ujal.

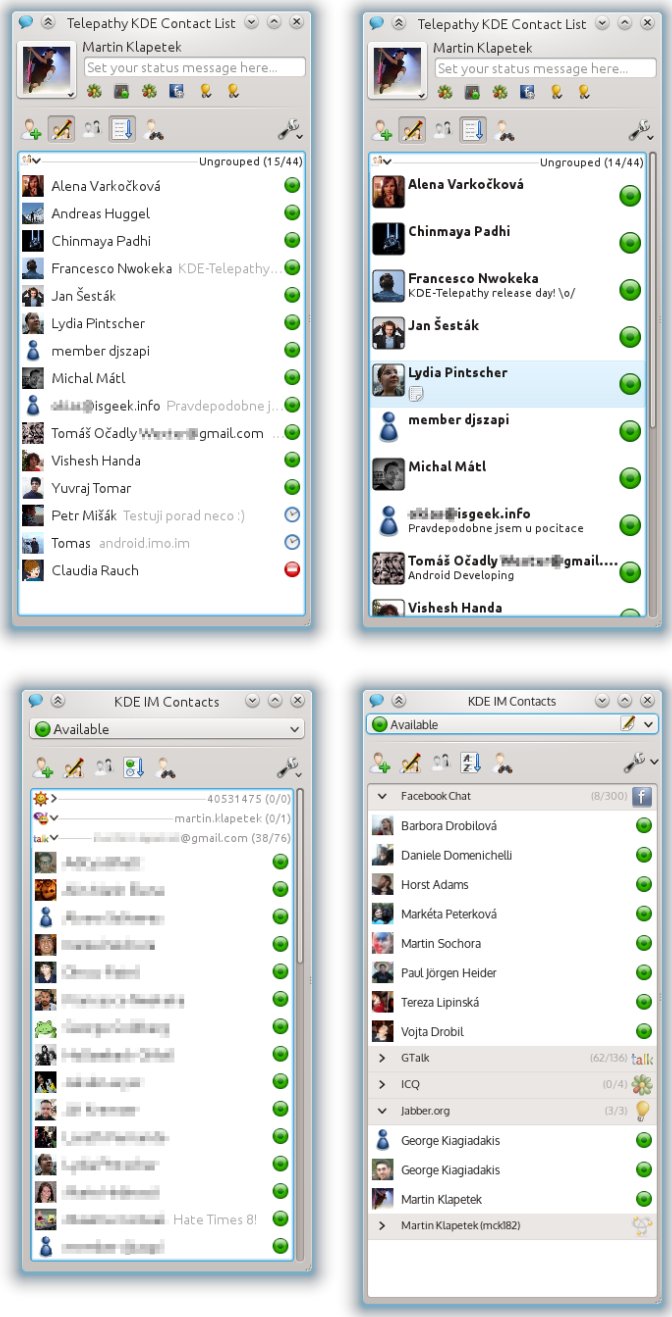
Pro testování dalších komponent se používal velmi jednoduchý nástroj, který zobrazoval jednoduchý seznam kontaktů a umožňoval pouze zahájit chat. Nástroj nebyl nikdy zamýšlen pro opravdové použití a proto jeho kód byla spíše „splácanina“, jejíž části nebylo možné znovu použít jinde. Novou aplikaci jsem tedy navrhl tak, aby její jednotlivé komponenty bylo možné znovu použít na kterémkoliv jiném místě.

3.5.5.1 Architektura Aplikace se skládá v podstatě ze tří částí - nastavení IM stavu, nastavení pohledu a samotný pohled, jehož součástí jsou také delegáty vykreslující samotný obsah pohledu. Všechny části jsou plně znovu samostatně použitelné. Aplikace navíc nemá hlavní menu, nebot'by v něm byly pouze duplikovány položky z toolbaru pro nastavení pohledu, což nedává smysl. Aplikace dříve obsahovala i obrázek s avatarem aktivního účtu. Problém byl více aktivních účtů najednou - který avatar zobrazit? Původní řešení implementovalo rozbalovací nabídku, ze které si uživatel vybral, který avatar chce vidět. Ten pak také byl nastaven všem ostatním účtům. To představovalo další problém, protože některé sítě nastavování vlastních avatarů nepodporují (např. pro Facebook Chat je avatar pouze pro čtení). Ve verzi 0.3 jsme se pak rozhodli obrázek s avatarem odstranit úplně, protože přinášel více problémů než řešil.

Veškeré ovládací prvky pro zahájení akcí s kontakty, o kterých se píše níže, jsou připojeny ke slotům, které z aktivovaného indexu získají účet, na kterém otevřou kanál podle vybraného typu akce. Žádost o nový kanál předá Account Manager Channel Dispatcheru a ten nastartuje aplikaci, která je schopná daný druh kanálu obsloužit. Žádosti je navíc možno předat i název preferovaného klienta, tím můžeme zaručit, že pokud má uživatel nainstalováno více klientů schopných obsloužit daný kanál, bude spuštěna aplikace z KDE Telepathy. Protože se implementace zahajování kanálů duplikovala na mnoha místech nejen v této aplikaci, byl kód přesunut do knihovny ktp-common-internals a vytvořeno rozhraní `KTp::Actions`, které implementuje vytvoření všech možných kanálů v jedné třídě.

Protože architektura Telepathy umožňuje udržovat spojení do IM sítí bez nutnosti uživatelského rozhraní, nemá aplikace pro seznam kontaktů koncept „skrýt do systémové části panelu“. Jako náhrada za ikonu v systrayi slouží plasmoid pro ovládání statusu, více v kapitole 3.5.7.1. Navíc modely i pohled se inicializují velmi rychle, není tedy potřeba udržovat aplikaci v paměti a proto se po zavření okna aplikace skutečně ukončí.

3.5.5.2 Nastavení IM stavu Telepathy podporuje 6 základních nastavitelných stavů - Přítomen (Available), Zaneprázdněn/Nerušit (Busy/Do Not Disturb), Pryč (Away), Nedostupný (Not Available), Neviditelný (Invisible) a Odpojen (Offline). Pro každý stav kromě Odpojen je možné nastavit také vlastní zprávu.



Obrázek 5: Vývoj aplikace, zleva nahoře: verze 0.1 s kompaktním delegátem, verze 0.1 s normálním delegátem, verze 0.3 s kompaktním delegátem a odstraněnými prvky pro nastavení avataru a jednotlivých statusů, verze 0.6 s vylepšeným kompaktním delegátem

První verze aplikace obsahovala ovládací prvky pro každý účet zvlášť a jeden společný prvek pro nastavení vlastní zprávy statusu. Toto bylo realizováno speciálním tlačítkem (`QToolButton`), které po kliknutí zobrazilo menu s jednotlivými stavy. Toto menu navíc obsahovalo i pole pro vlastní zprávu statusu. To umožňovalo nastavit pro každý účet jinou zprávu. Různé zprávy statusu pak bylo možné sjednotit použitím společného pole.

Vzhledem k tomu, že účelem KDE Telepathy je nabídnout komunikačního klienta, který v sobě stírá rozdíly mezi jednotlivými IM sítěmi, lze bezpečně předpokládat, že velké procento uživatelů bude mít alespoň dva a více aktivních účtů. Nastavovat v této situaci stav pro každý účet zvlášť by bylo poměrně nepohodlné, což po vydání první verze potvrdili i samotní uživatelé. Začal jsem proto pracovat na novém, jednotném způsobu nastavení stavu pro všechny uživatelovy účty.

Tento způsob nahradil jednotlivá tlačítka jedním comboboxem, který po rozbalení nabízel standardní statusy. Vlastní statusová zpráva byla realizována přeměnou comboboxu na textové pole. Uživatel tedy nejprve vybral základní status pomocí rozbalovacího tlačítka comboboxu a poté kliknutím na combobox mohl zadat vlastní zprávu. Toto chování nebylo ale úplně ideální, protože rozbalovací tlačítko je poměrně malé a standardní chování comboboxů po kliknutí přímo na něj zobrazí menu. Proto na combobox bylo umístěno nové tlačítko, které po kliknutí přemění combobox na textové pole. Samotné kliknutí na combobox pak zobrazilo menu, jak tomu je i v jiných aplikacích.

Jednotné nastavení statusu ale představuje problém v situaci, kdy má uživatel povoleno více účtů, z nichž alespoň jeden nepodporuje všechny statusy. Takovým účtem je např. účet technologie Salut (link-local XMPP), jehož specifikace definuje pouze stavy „Přítomen“, „Pryč“ a „Nerušit“. Vyvstává tedy otázka, co se má stát, pokud uživatel zvolí např. stav „Nedostupný“ a má povolenou kombinaci účtů GTalk a Salut. V podstatě existují čtyři možnosti, jak tento problém vyřešit. První možnost je zobrazit uživatelem vybraný status (tedy „Nedostupný“), nastavit tento status na účtech, které jej podporují a zbylé účty prostě nechat na jejich aktuálním statusu. Toto řešení jsme ale zamítli, protože podává uživateli nepravdivé informace - uživatel je v domnění, že všechny účty jsou „Nedostupné“, což není pravda. Druhá možnost je rozšíření první možnosti - zobrazit speciální status, řekněme „Vlastní“, který se zobrazí pokaždé, když účty nemají stejný status. To vyžaduje dodatečný prvek v uživatelském rozhraní, který zobrazuje stav jednotlivých účtů. Třetí možnost je vytvořit průnik podporovaných statusů všech povolených účtů a v comboboxu zobrazovat pouze tyto. Tato možnost nebyla vyloženě zamítnuta, ale nebyla nikdy realizována ze dvou důvodů - získat seznam podporovaných statusů daného účtu není jednoduché a navíc podmínečné skrývání ovládacích prvků není příliš dobré pro použitelnost; bylo by potřeba uživateli vysvětlit, proč po povolení nějakého účtu polovina statusu prostě zmizela. Poslední možnost je kombinace druhé a třetí možnosti - na všech účtech se nastaví požadovaný status. Podle specifikace se Account Manager pokusí nastavit co nejbližší status. Po nastavení se tedy projdou jednotlivé účty a zjistí se, který účet je „nejvíce online“. Pokud nalezený „nejvíce online“ status není shodný s uživatelem nastaveným statusem, nastaví se tento nový status všem účtům a situace se opakuje, dokud nemají všechny účty stejný status. Tento status je pak zobrazen uživateli.

Nakonec byla implementována právě poslední metoda. Byla vybrána proto, že jsme

chtěli skutečný globální status. Ukazuje se ale, že tento postup není úplně nejlepší a zcela selhává, pokud má uživatel aktivní SIP účet, neboťty podporují pouze stavy „Dostupný“ a „Odpojen“. Do verze 0.7 máme v plánu tuto situaci vyřešit poskytnutím kombinace globálního statusu a statusů pro jednotlivé účty.

Globální status je implementován třídou `GlobalPresence`, která byla později přesunuta do knihovny ktp-common-internals. V API této třídy jsou dvě důležité metody - `KTp::Presence currentPresence()` a `setPresence(const Tp::Presence &presence)`. První zmíněná metoda vrací aktuální globální status. Druhá metoda nastaví všechny povolené účty na status předaný jako argument. Toho je dosaženo jednoduchou iterací nad účty:

```
Q_FOREACH(const Tp::AccountPtr &account, m_enabledAccounts->accounts()) {
    account->setRequestedPresence(presence);
}
```

Jakmile Account Manager provede změnu, obdržíme signál `currentPresenceChanged()` prostřednictvím objektu `Tp::Account`, které `GlobalPresence` monitoruje. Slot `onCurrentPresenceChanged()` pak provádí postup popsaný výše - projde všechny účty, získá status, který je nejvíce online, pokud se liší od uloženého aktuálního statusu, je vyslán signál `currentPresenceChanged()` s novým statusem předaným jako parametr a tento status je také uložen. V této fázi se stavy účtů nacházejí v nekonzistentním stavu a o jejich sjednocení se momentálně stará uživatelské rozhraní. V našem seznamu kontaktů signal `currentPresenceChanged()` vyvolá změnu v comboboxu, který odráží aktuální status. Tato vyvolaná změna obratem na všech účtech nastaví nový status. Pokud je požadovaný status stejný jako aktuální status, Account Manager neprovede žádnou změnu. Tento mechanismus ovšem spoléhá na to, že aplikace seznam kontaktů je spuštěna, což vzhledem k architektuře není ideální řešení. Proto bylo navrženo přesunutí tohoto algoritmu do samotné třídy `GlobalPresence`.

Uživatel má navíc možnost si přednastavit vlastní statusy s vlastní zprávou. Rozbalený combobox má jako úplně poslední položku „Nastavit vlastní statusy“, po jejímž vybrání se otevře dialogové okno, kde lze spravovat vlastní statusy, které se pak objeví v nabídce statusů po rozbalení comboboxu. Seznam vlastních statusů je uložen v konfiguračním souboru aplikace.

3.5.5.3 Pohled a delegáty

3.5.5.3.1 ContactListWidget Třída `ContactListWidget` dědí ze standardní třídy pro stromové pohledy `QTreeView`. `ContactListWidget` v sobě zapouzdřuje vše potřebné pro jednoduché znovupoužití. Konstruktor provádí veškerou inicializaci - nejprve vytvoří modely (`ContactsModel` a `ContactsModelFilter`) a delegáty, poté provede základní nastavení pohledu jako vypnutí odsazení při vykreslování jednotlivých větví modelu či deaktivaci standardní interakce s myší, kterou pohled implementuje sám. Dále se načte uložený konfigurační soubor pohledu a provedou se další nastavení podle uložených hodnot - má-li seznam zobrazovat odpojené uživatele, který delegát

se má použít atd. Konfigurace se ukládá s každým zavřením aplikace. Samotný pohled zůstává nečinný, dokud mu není předána instance `Tp::AccountManager`. Tuto instanci dále předá modelu, čímž nastartuje jeho naplnění kontakty. K předání instance `Tp::AccountManager` se používá samostatná metoda a nikoliv konstruktor, protože sám `Tp::AccountManager` se inicializuje asynchronně. Zatímco se tak čeká na připravení objektu, připraví se uživatelské rozhraní. Uživatel tedy vidí plně spuštěnou aplikaci, která načítá data.

Veřejné API třídy `ContactListWidget` poskytuje způsob, kterým lze pohled nastavit „zvenčí“. Podoba ovládacích prvků je tak čistě na vývojáři aplikace, která pohled používá. V naší aplikaci jsme použili kombinaci toolbaru a rozbalovací menu, implementovaných ve třídě `MainWidget`. Toolbar obsahuje akce, ke kterým je potřeba častý nebo rychlý přístup. Najdeme tam tak akce pro přepnutí mezi setříděním kontaktů podle skupin či podle účtů, zobrazení či skrytí odpojených uživatelů, seřazení kontaktů podle názvu nebo podle statusu, tlačítko pro vyhledávání v seznamu a nakonec ještě tlačítko pro přidání nových kontaktů. Poslední tlačítko, odsazené úplně vpravo, je tlačítko s rozbalovacím menu, které ukrývá všechny ostatní volby a možnosti, jako je typ seznamu (typ použitého delegátu), typ zobrazených kontaktů (všechny kontakty, pouze neblokované, pouze blokované), dále poskytuje přístup do nastavení účtů a nastavení KDED modulu (otevře příslušné KCM moduly), možnost vytočit libovolné číslo pro zahájení hovoru a možnost připojení k chatovací místnosti (která je definována v rozšíření XMPP XEP-0045).

Do budoucna bychom se rádi celého toolbaru zbavili a tlačítko s rozbalovacím menu umístili vedle ovládání statusu. Všechny akce z toolbaru by byly přesunuty do jeho menu a přidány klávesové zkratky pro rychlý přístup. Dále by měla přibýt funkce „vyhledávání psaním“, tedy pokud je aplikace aktivním oknem, psaní na klávesnici by mělo začít vyhledávat v seznamu. Díky návrhu jednotlivých částí aplikace si tato změna vyžádá minimální zásah do kódu.

`ContactsListWidget` neimplementuje kontextovou nabídku pro kontakty. O tu se stará nadřazený widget `MainWidget`, který naslouchá signálům pohledu o žádost kontextové nabídky při kliku pravým tlačítkem myši a reaguje na ně zobrazením nabídky na pozici myši. Kontextová nabídka samotná je implementována třídou `ContextMenu` a její fungování je velmi jednoduché - pohled v signálu předá index kontaktu, na který uživatel klikl pravým tlačítkem a podle toho sestrojí vlastní menu. Menu obsahuje akce, které je s kontaktem možné zahájit (chat, audio nebo video hovor) a akce jako smazání kontaktu nebo přesun do skupiny. Veškeré aktivované akce zpracovává samotná třída `ContextMenu`. I tato část aplikace je tedy plně samostatná a může být jednoduše znovu použitá.

Kromě kontextového menu se `MainWidget` ještě stará o vyskakovací „bubliny“ (tooltips) nad kontakty, které se zobrazí po najetí myší na kontakt. Původně byla využita standardní implementace bublin z Qt, která dovolovala HTML formátování. To ale nebylo dostatečně flexibilní a dlouhý HTML kód uprostřed C++ kódu působil zmatečně. Proto jsme později přešli na řešení z jiných KDE aplikací, které využívají bubliny založené na QWidgets, což znamená plně flexibilní a přizpůsobitelné řešení bez nutnosti HTML.

Princip fungování je pak stejný jako u kontextové nabídky - pohled signalizuje žádost o tooltip a předá index, nad kterým je právě ukazatel myši, z toho se získají všechna potřebná data a tooltip se vykreslí na obrazovku. Po přesunu myši mimo kontakt bublina samozřejmě zmizí.

3.5.5.3.2 AbstractContactDelegate Tato třída implementuje základ pro delegáty použité v naší aplikaci. Při rozšiřování standardních Qt delegátů je potřeba reimplementovat dvě metody `paint()` a `sizeHint()`. Metoda `paint()` provádí vlastní vykreslování pomocí předaného objektu `QPainter`. Druhým argumentem funkce je struktura obsahující informace o aktuálním systémovém stylu (velikost písma, paleta barev atd.) a také rámec, do kterého se bude vykreslovat. Poslední parametr je aktuálně zpracovávaný index, ze kterého získáme data, která chceme vykreslit. Druhá metoda `sizeHint()` jednoduše vrací velikost rámce, do kterého se má kreslit. I této funkci je předána struktura s parametry aktuálního stylu a index. Díky tomu může `sizeHint()` rozlišovat různé prvky a vracet pro ně různé velikosti. Přesně toho využívá náš abstraktní delegát, který vrací jinou velikost vykreslovacího rámce pro seskupující prvek (skupina nebo účet; metoda `sizeHintHeader()`) a jinou velikost pro prvek kontaktu (čistě virtuální metoda `sizeHintContact()`). Všechny implementace `sizeHint()` metod vypočítávají velikost na základě uživatelovy nastavené velikosti písma v systému. Tím je zajištěn správný vzhled i při velmi vysokých rozlišeních, kde písma mají daleko větší DPI.

Reimplementovaná metoda `paint()` provádí úplně to samé - podle typu zpracovávaného prvku zavolá buď metodu `paintContact()` nebo `paintHeader()`. Metoda `paintContact()` je čistě virtuální a reimplementují ji rozšiřující třídy. Metodu `paintHeader()` implementuje přímo třída `AbstractContactDelegate` a to proto, že oba delegáty v naší aplikaci vykreslují seskupující položku stejně, tím se předchází duplikaci kódu.

Pro verzi 0.6 jsem se rozhodl změnit podobu seskupujících prvků, aby byly více konzistentní s podobou prvků kontaktů. Změna prošla několika iteracemi a diskuzí s týmem specializujícím se na použitelnost uživatelských rozhraní. Výsledek je na obrázku 5.

3.5.5.3.3 ContactDelegate `ContactDelegate` byl vizuálně navržen na setkání týmu v září roku 2010, kdy byl celý projekt ve svých počátcích. Části pak byly implementovány krátce po setkání, kvůli chybějícím modelům byla ale práce brzy zastavena. Navázal jsem na ni v únoru roku 2011 a dokončil podle původního návrhu. Delegát vykresluje kontakty s velkým avatarem nalevo, ikonou statusu napravo a dvěma řádky textu uprostřed. První řádek bylo jméno kontaktu, druhý řádek pak zpráva statusu, pokud měl kontakt nějakou nastavenou. Po najetí myší na kontakt se zpráva statusu zmizením prolnula s tlačítky, které poskytovaly přístup k možným akcím kontaktu - zahájení chatu, audio hovoru, video hovoru nebo sdílení plochy. Tlačítka se prolnula zpět se zprávou, pokud se kurzor myši dostal mimo položku kontaktu. Jednotlivá tlačítka zobrazovala po najetí myší na ně krátký popisek, co vlastně dělají. Později se do Telepathy-Qt dostala také podpora typu klienta, který kontakt používá - počítač nebo mobilní zařízení. Je-li uživatel k IM

síti přihlášen prostřednictvím mobilního zařízení, indikuje to ikona mobilního telefonu vedle ikony statusu.

3.5.5.3.4 ContactDelegateCompact Druhým a výchozím delegátem je `ContactDelegateCompact`. Jak už název třídy naznačuje, jedná se o kompaktnější verzi původního delegátu, oproti kterému má menší ikony a vše vykresluje v jednom řádku, jak je vidět na obrázku 5 na straně 29.

V diskuzi s uživateli na výroční konferenci KDE, Akademy, nám bylo navrženo přidání třetího delegátu, který by byl velmi minimalistický a umožnil zobrazení co největšího počtu kontaktů v pohledu. Implementace takového delegátu by byla shodná s kompaktním delegátem, pouze by se změnil `sizeHint()`, velikost ikon a velikost písma. Rozhodli jsme se proto zkusit delegát implementovat. Aby se předešlo zbytečné duplikaci kódu, byl kompaktní delegát rozšířen o konfigurovatelné hodnoty velikostí písem a ikon a přidána veřejná metoda pro přepnutí mezi kompaktním a minimalistickým módem. Metoda pouze změnila hodnoty proměnných ukládající zmíněné parametry podle požadovaného módu. Vykreslovací funkce pak zůstala beze změn. Za necelých 10 minut bylo hotovo a my jsme mohli uživatelům nový delegát rovnou předvést. Protože jsme byli spokojeni jak my, tak uživatelé, tento minimalistický delegát jsme začlenili jako novou funkci.

3.5.6 Aplikace obsluhující audio/video hovory

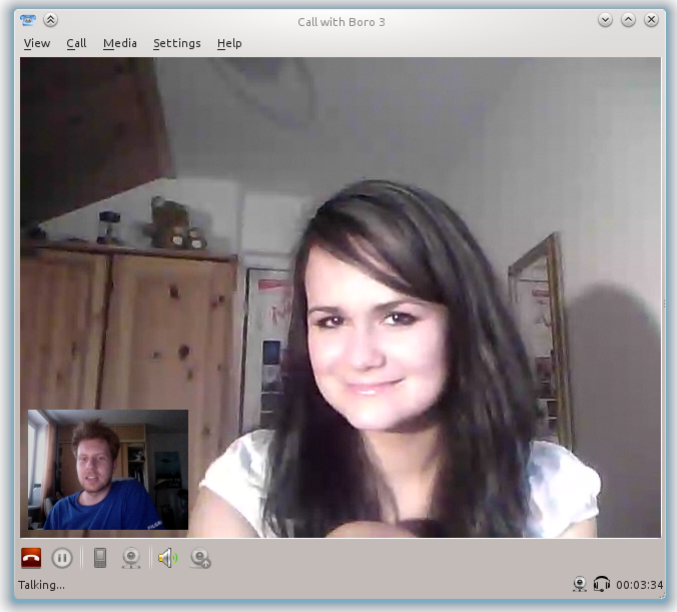
Klient obsluhující kanály s audio a video hovory byl do KDE Telepathy pod názvem Call UI přidán až ve verzi 0.4. To především proto, že se čekalo na dokončení specifikace Telepathy Call1, která značně vylepšila možnosti audio/video hovorů v Telepathy.

Audio a video hovory v Telepathy jsou realizovány pomocí knihovny Farstream, za kterou stojí také společnost Collabora Ltd. Farstream je kompletní multi-platformní framework pro audio a video komunikaci, včetně více-uživatelských konferencí. Je stejně jako Telepathy modulární a založen na široce používaném multimediálním frameworku GStreamer. Farstream spolu s Telepathy byli první implementací multimediálního XMPP rozšíření Jingle [7]. Díky modulárnosti Farstreamu je velmi jednoduché napsat plugin pro jakýkoliv nový protokol. GStreamer pak poskytuje potřebné kodeky a další nízkoúrovňové rutiny potřebné pro kódování a dekódování multimediálních proudů.

Call UI je rozdělen na knihovnu, která obsluhuje vlastní multimediální kanál a aplikační část, která vytváří uživatelské rozhraní a slouží jako Telepathy klient, kterému Channel Dispatcher deleguje kanály. Celou aplikaci je možné sestavit pouze pokud systém obsahuje knihovnu Telepathy-Qt sestavenou s podporou Telepathy-Farstream.

Aplikace i uživatelské rozhraní jsou poměrně jednoduché. Pokud je kanál video hovorem, zobrazuje aplikace přijímaný video proud a v něm obraz lokální kamery, pokud uživatel nějakou má. V případě, že kontakt neposílá video proud, nebo pokud se jedná pouze o audio hovor, je místo videa zobrazen avatar kontaktu.

Původní vývojář komponenty byl George Kiagiadakis, jehož práci sponzorovala Collabora. S nedávným posunem priorit Collabory ovšem zcela ustal i vývoj této komponenty. Ta tak nabízí pouze základní funkce a aktuálně se pro ní hledá nový vývojář.

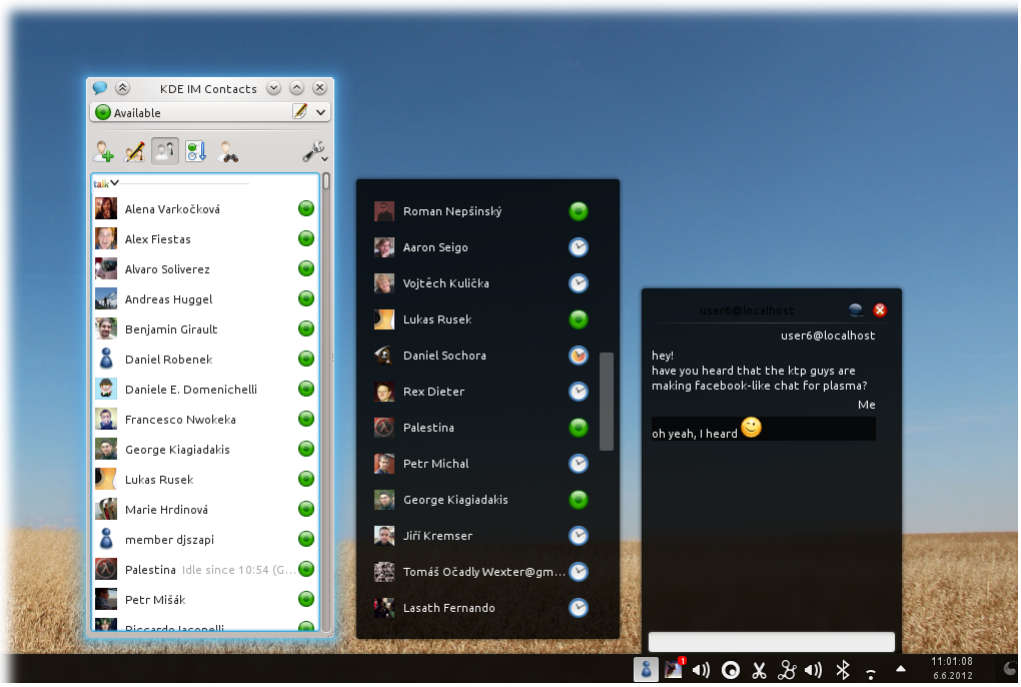


Obrázek 6: Aplikace Call UI

3.5.7 Applety pro integraci s pracovním prostředím

Naším primárním cílem jsou uživatelé pracovních prostředí KDE Plasma Workspaces. Plasma, coby framework pro grafická pracovní prostředí, umožňuje tvorbu jednoduchých i komplexnějších nástrojů a jednoúčelových aplikací, které přebírají její vzhled a chování. Vizuálně se tedy liší od běžných aplikací a působí jako přímá součást pracovního prostředí. Těmto miniaplikacím se v terminologii Plasmy říká widgety, applety, nebo také plasmoidy. Po příchodu Qt Quick a jazyka QML se Plasma vydala plně vstříc těmto technologiím, které přináší lepší vzhled, vyšší výkon a menší potřebu údržby díky nahrazení tisíce řádků C++ kódu za pár stovek řádků QML, přičemž veškerá funkcionalita je plně zachována. Plasma klade velký důraz na oddělení prezentační a datové logiky. O datovou logiku se starají DataEnginy - standardizované rozhraní pro poskytování dat prezentační vrstvě. Tyto DataEnginy jsou sdíleny a jejich poskytovaných dat může využít kterýkoliv plasmoid. Pro QML plasmoidy je možné použít ještě druhý způsob poskytování dat - tzv. QML pluginy, které jsou psány v C++/Qt a využívají Qt Meta-Object systému.

3.5.7.1 Plasmoid pro status Jako první vznikl plasmoid zobrazující aktuální status. Byl napsán už v roce 2008, tehdy ještě pro projekt Decibel. Od té doby prošel několika výraznými přepisy. První byl po zániku Decibelu a vzniku KDE Telepathy, druhý byl s příchodem QML a zatím poslední přepis byl, když už se plasmoid stal natolik kom-



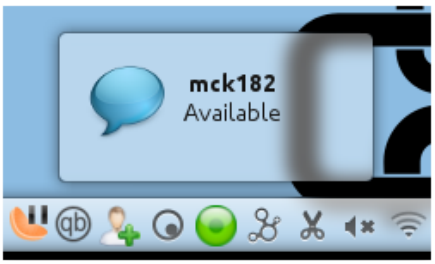
Obrázek 7: Zleva: Aplikace pro seznam kontaktů, plasmoid pro seznam kontaktů a plasmoid pro chat

plikovaným a plným chyb, že jsme se rozhodli začít znovu. Původní plasmoid využíval DataEngine pro komunikaci s Telepathy, to ale dělalo architekturu plasmoidu daleko složitější, než bylo opravdu potřeba. DataEngine jsme tedy zahodili a celý plasmoid zjednodušili na pouhou ikonu s kontextovým menu. Aby byl kód co nejjednodušší, vrátili jsme se zpět od QML k standardním appletům psaným v C++/Qt a vše implementovali v jedné třídě. Kód je velmi jednoduchý a přímočarý, celkem má 380 řádků (bez hlavičkového souboru).

Plasmoid zobrazuje globální status a kontextovým menu umožňuje rychle změnit status na jiný. Do kontextové nabídky později přibyly i další možnosti jako otevření nastavení účtů, připojení k chatovací místnosti či vytočení čísla pro audio/video hovor. Levým kliknutím na plasmoid se pak spustí aplikace seznam kontaktů. Pokud aplikace již běží, ale není právě aktivní aplikací, je její okno přeneseno dopředu. Je-li aktivní aplikací, kliknutí na plasmoid aplikaci ukončí.

Díky designu Plasmy je možné plasmoid umístit kamkoliv do panelu, na plochu či do systémové části panelu (systray) a je dokonce možné mít více než jeden, což se hodí, pokud máme třeba více obrazovek s více panely. Plasmoid je v architektuře KDE Telepathy plně samostatná komponenta a jejím přidáním do systémové části panelu lze napodobit klasické monolitické IM klienty a přitom si stále zachovat plnou nezávislost na ostatních komponentách.

Podobně jako v aplikaci seznam kontaktů, i u tohoto plasmoidu máme v plánu rozšířit



Obrázek 8: Plasmoid pro status umístěn v systémové části panelu se zobrazenou info bublinou

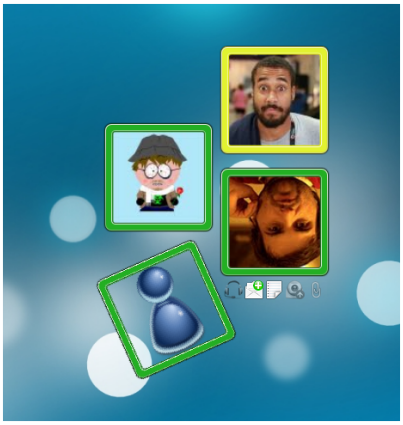
funkčnost na kombinaci ovládání globálního statusu a ovládání statusů jednotlivých účtů zvlášť.

3.5.7.2 Seznam kontaktů v QML Tento velmi jednoduchý applet vznikl původně jako jednoduchá technická ukázka architektury KDE Telepathy a jednoduchosti znovupoužití našich modelů i s naprosto jiným uživatelským rozhraním (napsaným v jiném jazyce). Plasmoid nenabízel vůbec žádné funkce, pouze zobrazoval neseřazený seznam kontaktů a po kliknutí na kontakt se zahájil chat. I tak jsme se rozhodli plasmoid zařadit do stabilního vydání, jednak abychom ukázali, kam chceme projekt směřovat, a jednak abychom získali zpětnou vazbu uživatelů. Ta byla velmi pozitivní a dostali jsme zpět i spoustu nápadů na vylepšení a nové funkce. Aktuálně je jedním ze zamýšlených použití přidání plasmoidu do panelu, který se automaticky schovává (např. na straně obrazovky). Uživatel tak má seznam kontaktů neustále okamžitě k dispozici po pouhém najetí na hranu obrazovky.

3.5.7.3 Plasmoid pro jednotlivé kontakty V rámci programu Google Summer of Code vznikl také plasmoid, který reprezentuje jeden kontakt. S jeho použitím si tak uživatel může přidat svoje oblíbené nebo důležité kontakty na plochu či do panelu a mít rychlý způsob, jak s konkrétním kontaktem zahájit chat nebo audio/video hovor. Status daného kontaktu reprezentoval barevný rámeček kolem jeho avataru - zelená reprezentuje status „Dostupný“, žlutá „Pryč“, červená „Zaneprázdněn“ a šedá pak „Odpojen“.

Kontakt bylo možné přidat buďto standardní cestou, kterou se přidávají plasmoidy, kdy se zobrazil uživateli seznam kontaktů a uživatel vybral, který kontakt v plasmoidu chce, nebo pak přetažením kontaktu z aplikace pro seznam kontaktů.

3.5.7.4 Plasmoid pro chat Tento plasmoid byl původně zamýšlen pouze jako rozšíření standardních systémových notifikací (které jsou implementovány také jako plasmoid) o možnost okamžité odpovědi přímo z notifikace. Tedy uživatel obdrží novou zprávu - zobrazí se notifikace - uživatel najede myší na notifikaci - zobrazí se vstupní textové pole - uživatel klikne a začne psát - klávesou Enter pak zprávu odešle. Pro tento nápad



Obrázek 9: Několik plasmoidů pro kontakty

byly inspirací notifikace z GNOME 3. Při konkrétním návrhu pak padla myšlenka, že by se notifikace mohla rozšířit na jednoduché chatovací rozhraní. A protože standardní notifikace v prostředí Plasma se zobrazují „připnuté“ k panelu, rozhodli jsme se celý návrh proměnit v chatovací rozhraní známé z webů jako GMail nebo Facebook. Většina uživatelů má na obrazovce neustále viditelný panel a přenesením chatu do tohoto panelu by odpadla potřeba přepínat aplikace, protože uživatel by měl aktivní chat dostupný neustále na obrazovce. Dokonce i reagováním na zprávy by uživatel nemusel vůbec opouštět svou aktivní aplikaci.

První verze implementována v QML s pomocí QML pluginů byla vydána s verzí 0.4. Plasmoid byl stejně jako plasmoid pro seznam kontaktů uveden jako technická ukázka a ukázka směru, kam se projekt vydává. Během verzí 0.5 a 0.6 byla značná část plasmoidu (včetně pluginů) podstatně přepsána a vylepšena.

3.5.8 Aplikace zajišťující přenos souborů a aplikace umožňující posílání souborů z jakékoliv aplikace

Příchozí kanál pro přenos souborů nejprve přijme schvalující komponenta. Ta zobrazí uživateli zprávu o novém příchozím kanálu, kterým je v tomto případě přenos souboru, a nabídne mu jeho přijetí. Pokud uživatel přijme, schvalující komponenta kanál potvrdí a Channel Dispatcher aktivuje klienta, který je schopný kanál pro přenos souborů obsloužit.

Náš klient obsluhující příjem souborů neposkytuje žádné uživatelské rozhraní, přenos probíhá na pozadí (zajišťuje jej Connection Manager) a o průběhu přenosu je uživatel informován prostřednictvím systémové komponenty zobrazující průběh operací jako kopírování či přesun souborů (příjem souboru je v podstatě pouhé kopírování souboru). Z tohoto místa je také možné přenos přerušit. Pokud již cílový soubor existuje, je uživatel pochopitelně vyzván k zadání nového jména souboru. Po přijetí souboru je zobrazena systémová notifikace s tlačítkem pro přímé otevření souboru.

Architektura klienta je poměrně jednoduchá - po předání kanálu se zjistí, zda-li se jedná o příchozí či odchozí kanál a podle toho se nastartuje příslušná úloha (job). Obě úlohy v podstatě pouze procházejí jednotlivými kroky přenosu souboru a reagují na změny stavu kanálu.

Pro odesílání souborů jsme vytvořili jednoduchý nástroj ktp-send-file, který se standardně instaluje jako plugin do správce souborů Dolphin (výchozí správce souborů pro KDE Plasma Desktop). V Dolphinu uživatel klikne pravým tlačítkem na libovolný soubor a v kontextové nabídce vybere „Odeslat soubor IM kontaktu“. Tato akce aktivuje dialog pro výběr kontaktu a po zvolení se vytvoří nový kanál pro přenos souborů, který Channel Dispatcher deleguje komponentě popsané výše. Ktp-send-file lze použít z jakékoliv aplikace, stačí při spuštění předat cestu k souboru jako parametr. Pokud žádný parametr není předán, ktp-send-file nejprve zobrazí dialog pro výběr souboru.

Obrázek 10: Odesílání souborů z Dolphinu pomocí nástroje ktp-send-file

3.5.9 Modul pro KDED pro integraci se systémem

Jak již bylo zmíněno v kapitole 3.5.3, při startu prostředí KDE Plasma se spouští proces nazvaný KDED, který běží na pozadí po celou dobu, kdy je uživatel přihlášen v pracovním prostředí. KDED v pracovním prostředí vykonává na pozadí různé úlohy, které jsou implementovány v jednotlivých modulech, které KDED spouští a sdružuje do jednoho jediného procesu. Je proto kriticky důležité, aby žádný z modulů nevykonával blokující volání, které by zastavilo celý proces při čekání na návrat z funkce. V nastavení pro KDED je navíc možné jednotlivé moduly spustit či zastavit a vidět jejich aktuální stav.

Pro KDE Telepathy jsem chtěl implementovat funkce jako automatické nastavení statusu na „Pryč“ po určité době či integraci s hudebními přehrávači v podobě automatického nastavení zprávy statusu na aktuálně poslouchanou skladbu. Protože uživatel může být prostřednictvím Telepathy připojen k IM sítím a nemít přitom spuštěného klienta či jinou aplikaci, která by výše zmíněné funkce obstarávala, potřeboval jsem vlastní proces, který by běžel po celou dobu přihlášení. Realizace tohoto procesu jako KDED modulu byla tedy jasná volba. Díky tomu navíc odpadla starost o automatické spouštění procesu po přihlášení, starost o náhlé ukončení procesu (např. uživatelem) atp. KDED toto všechno obstarává sám a jeho běh (včetně obnovení po pádu) zajišťuje proces kdeinit, jehož ukončením dojde k odhlášení uživatele.

Každý KDED modul musí rozšiřovat třídu KDEDModule a podobně jako KCM modul, i KDED modul musí být exportován (i zde lze použít `KPluginFactory`). Představa byla taková, že bude existovat několik jednoduchých, jednoúčelných pluginů, které by modul načítal. Plugin by měl dva signály - jedním by signalizoval, že byl aktivován a druhým by signalizoval požadovanou změnu statusu. Původní návrh architektury počítal i s prioritami pluginů, kdy by plugin s nižší prioritou, než aktuálně aktivní plugin, nemohl změnit status. Priority byly i implementovány, systém pluginů ale nikdy nebyl realizován jako skutečný systém pluginů (pro pouhé 2 pluginy to nemělo cenu) a hlavní modul pouze přímo vytvářel instance jednotlivých pluginů (načítání tedy bylo přímo „zadrátováno“ v kódu), priority tak nahradilo jednoduše pořadí, ve kterém byla instance pluginu přidána do seznamu pluginů, který udržuje hlavní třída modulu.

Samotný modul, implementovaný ve třídě TelepathyModule, pouze inicializuje Account Manager (v podobě objektu `Tp::AccountManager`), vytvoří instance všech pluginů a naslouchá signálu `activated(bool)`. Pokud některý plugin signál vyšle, projde se seznam pluginů (první v seznamu znamená nejvyšší prioritu, poslední nejnižší), ověří se, že je plugin aktivní a pokud ano, nastaví se status na jeho plugin. Jinými slovy se nastaví status z prvního aktivního pluginu v seznamu.

Pluginy rozšiřují třídu `TelepathyKDEDModulePlugin`, ve které je čistě virtuální metoda `QString pluginName()`. Tu musí rozšiřující třídy implementovat a vracet v ní vlastní název pluginu. Další dvě veřejné třídy - `bool isActive()` a `bool isEnabled()` vrací `true`, pokud je plugin aktivní (má připraven status k použití), resp. pokud je plugin vůbec povolen (může být vypnut v konfiguračním rozhraní). Dále má rozhraní tři chráněné (protected) metody - `setActive(bool)`, `setEnabled(bool)`, `setRequestedPresence(Tp::Presence)`. Metody jsou chráněny, protože hodnoty, které tyto metody mění, nesmí měnit jiný objekt než plugin samotný. Nakonec má třída dva signály - `activate(bool)`, kterým signalizuje hlavnímu modulu, že byl plugin aktivován (resp. deaktivován) a jeho status má být použit jako hlavní status, a `requestPresenceChange(Tp::Presence)`, což je starší metoda, která dříve byla použita pro signalizaci nového statusu, který se má použít.

Toto API, stejně jako algoritmy v hlavním modulu, prošlo několika aktualizacemi a i přes svoji plnou funkčnost by zasloužilo kompletní přepis. Ten je připravován do verze 0.8.

Aktuálně má modul dva pluginy - automatické nastavení statusu „Pryč“ a „Nedo-

stupný“ po nastavených minutách nečinnosti a automatické nastavení statusu na právě poslouchanou skladbu z jakéhokoliv přehrávače, který podporuje MPRIS2 specifikaci.

Pro získání času nečinnosti uživatele slouží třída `KIdleTime`, které předáme čas, za který chceme být informováni, obratem získáme přidělené id. KIdleTime pak vysílá signál pro každý registrovaný čas a předává v něm id, které bylo přiděleno danému času. Aplikace využívající `KIdleTime` tak při obsluze každého signálu ověří, jestli předané id je shodné s dříve přiděleným id a pokud ano, znamená to, že uživatel byl nečinný po požadovaný čas. Náš plugin registruje dva časy - jeden pro „Pryč“ a druhý pro „Nedostupný“. Když obdrží signál s přiděleným id pro „Pryč“, zkontroluje, zda-li aktuální status je „Přítomen“ nebo „Zaneprázdněn“ a pouze tehdy se aktivuje a nastaví status na „Pryč“, v ostatních případech to nedává smysl (např. když má uživatel jako aktuální status „Neviditelný“, přepnutí do „Pryč“ by ho zviditelnilo). Pokud přijde signál s id pro „Nedostupný“, nastaví se status pouze pokud aktuální status je „Pryč“. To proto, že status „Nedostupný“ je druhý stupeň statusu „Pryč“, automaticky se tedy nastaví pouze tehdy, pokud aktuální status už je „Pryč“. Při první uživatelově aktivitě se plugin deaktivuje a hlavní modul projde všechny aktivní pluginy a nastaví status na první aktivní plugin. Pokud žádný není aktivní, nastaví se status, který měl uživatel nastaven před aktivací pluginu.

Plugin pro nastavení statusu na aktuálně přehrávanou skladbu je implementován ve třídě `TelepathyMPRIS`. MPRIS je D-Bus rozhraní, které implementují hudební přehrávače. Skrze MPRIS rozhraní můžeme jednak získat informace o aktuálním stavu přehrávače (zda-li přehrává, co přehrává, v jakém čase skladba právě je atd.) a jednak přehrávač samotný ovládat. Specifikace je ale bohužel poněkud vágní v tom, ze specifikuje pouze samotné rozhraní, už nepopisuje, jak má přehrávač rozhraní implementovat. Každý přehrávač podporující MPRIS se tak chová trochu odlišně, především pak v tom, kdy a jak signalizuje změnu skladby.

`TelepathyMPRIS` tedy naslouchá D-Bus signálům specifikace MPRIS a získává informace o aktuálně přehrávané skladbě, ze kterých sestavuje zprávu statusu a je-li tento plugin povolen, zprávu statusu nastavuje aktuálnímu statusu, ten se tedy nemění, pouze se mění jeho zpráva. Pokud je přehrávač zastaven, je plugin deaktivován, na což opět reaguje hlavní modul nastavením statusu z jiného aktivního pluginu či předchozího statusu.

K práci s D-Busem se využívá QtDBus modul, který pro D-Bus poskytuje API ve stylu Qt. Použití je jednoduché, ale v případě MPRIS pluginu poměrně nebezpečné - část QtDBus API vykonává blokující volání a protože vše běží v jednom procesu, použití těchto funkcí by blokovalo celý KDED. Qt dokumentace u některých metod bohužel nezmiňuje, že jde o blokující volání. Náš plugin několik takových volání zpočátku používal, protože se ale jednalo o jednoduché dotazy na D-Bus, blokování KDED procesu bylo téměř neznatelné. Problémy objevili až vývojáři přehrávače Amarok, který také implementuje MPRIS specifikaci. Při vývoji některých funkcí Amaroku vývojářům zamrzával na 5 vteřin systém. Ukázalo se, že je to kvůli synchronním voláním z našeho MPRIS pluginu. Veškerá D-Bus volání jsem tedy v pluginu přepsal na asynchronní a při té příležitosti také vylepšil samotný algoritmus získávání dat o skladbě.

Krom pluginů, které mění globální status, obsahuje modul ještě 4 další „pluginy“ - automatické připojení po přihlášení, přijímání žádostí o autorizaci kontaktů, notifikace pro změnu statusu kontaktu a notifikace pro chybová hlášení. Tyto 4 pluginy už jsou obyčejnými třídami dědícími z třídy `QObject` a nejsou přidávány do seznamu pluginů, fungují jako samostatné objekty. Implementace je většinou vcelku triviální - automatické připojení po přihlášení nedělá nic jiného, než iteruje přes uživatelovy účty a nastavuje status uložený při odhlášení, chybová hlášení monitorují všechny účty a v případě chyb zobrazují systémovou notifikaci se srozumitelným textem chyby (ktp-common-internals obsahuje slovník chyb, který má pro všechny možné chyby Telepathy definován řetězec, který je srozumitelný pro uživatele, řetězce jsou samozřejmě i přeloženy). Třída pro notifikace o změnách statusu kontaktů monitoruje všechny kontakty a ve výchozím stavu nedělá nic. Uživatel má možnost si pro jednotlivé kontakty nastavit vlastní upozornění - systémovou notifikací, zvukem, spuštěním vlastního programu atd., toto nastavení se provádí z kontextového menu kontaktu v aplikaci pro seznam kontaktů. Je možné také nastavit chování pro všechny kontakty. Pokud tedy kontakt změní status, zkontroluje se uživatelova konfigurace a provede se vybraná akce (o provedení se stará systémový framework KNotify). Poslední z těchto čtyř „pluginů“ se pak stará o příchozí žádosti o autorizaci, tedy pokud si nás nějaký kontakt chce přidat do svého seznamu. Nová příchozí žádost vytvoří v systémové části panelu StatusNotifierItem, který skrze své kontextové menu umožňuje žádost buďto přijmout nebo zamítnout. Každá další příchozí žádost je přidána do již existujícího objektu StatusNotifierItem. Pokud uživatel žádost potvrdí a pokud kontakt ještě není v jeho seznamu kontaktů, je obratem kontakt také požádán o autorizaci, po potvrzení druhé strany (může být automaticky implementováno v protokolu) je kontakt přidán do uživatelova seznamu kontaktů.

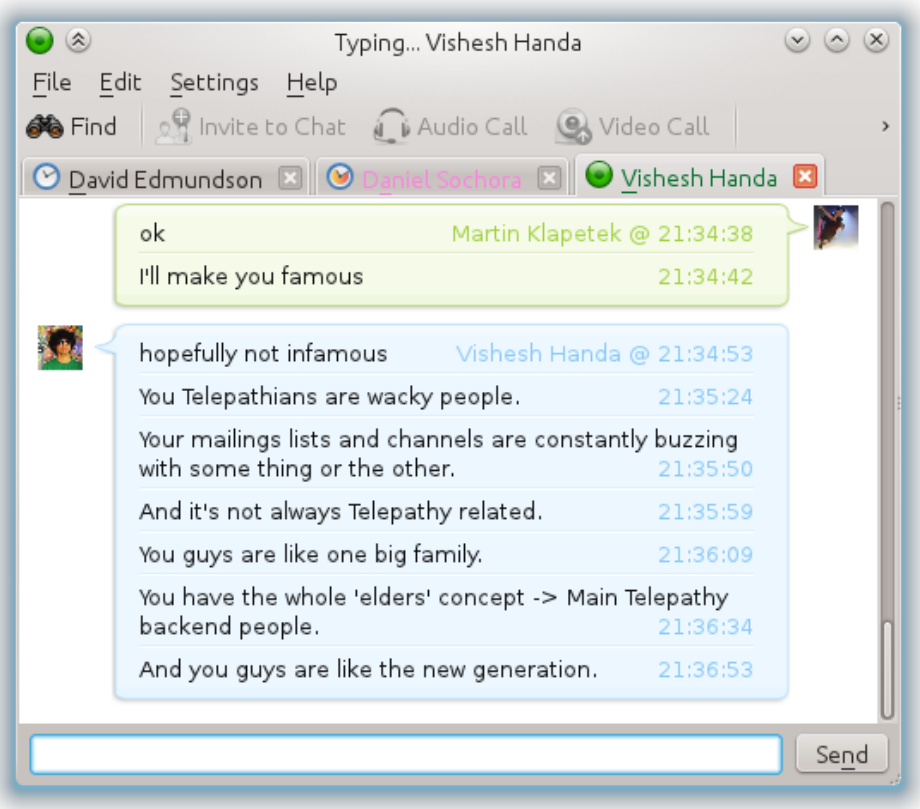
Na tomto je vidět, jak obrovskou výhodu „komunikace jako služba“ přináší - uživatel nemusí mít spuštěnou žádnou „viditelnou“ komponentu a přesto bude informován, když se jeho kamarád připojí či pokud se jej někdo pokusí přidat do svého seznamu.

Součástí KDED modulu je také konfigurační modul pro nastavení pluginů, který se stejně jako modul pro správu účtu instaluje do systémového nastavení, odkud je také přístupný.

3.5.10 Aplikace pro textový chat

Komponenta s názvem Text UI slouží jako klient obsluhující textové kanály. Protože je to komponenta, ve které stráví uživatel nejvíce času, byl při vizuálním návrhu kladen velký důraz na estetiku a jednoduchost aplikace. Na první pohled je aplikace velmi jednoduchá - skládá se pouze z nástrojové lišty s několika tlačítky, konverzačního pohledu a pole pro psaní zprávy.

Architektonicky je klient rozdělen na dvě části - sdílenou knihovnu, která poskytuje vlastní chatovací rozhraní, a aplikaci, která chatovací rozhraní vkládá do vlastního okna aplikace a slouží jako vlastní Telepathy klient, kanál k obsluze ale předává části v knihovně. Součástí repositáře je také aplikace pro prohlížení předešlých konverzací, která také využívá část sdílené knihovny.



Obrázek 11: Aplikace Text UI pro textový chat se třemi aktivními konverzacemi

Chatovací rozhraní bylo vytvořeno jako sdílená knihovna především z důvodů jednoduchého sdílení této komponenty. Jako příklad použití lze uvést kolaborativní editor či multiplayerovou hru, kde v obou případech chatovací rozhraní přijde velmi vhod.

Hlavní třídou knihovny je `ChatWidget`, který v sobě kombinuje všechny části knihovny v jeden ucelený widget. Ten se skládá z pohledu, kde se zobrazují zprávy konverzace, a ze vstupního pole, prostřednictvím kterého uživatel odesílá zprávy. Konverzační pohled třídy `AdiumThemeView` implementuje rozhraní, které je kompatibilní se styly populárního klienta Adium ze světa OS X. Pohled je ve skutečnosti HTML renderovací engine založený na třídě `KWebView` a Adium styly jsou kombinace HTML a CSS šablon. Toto řešení na jednu stranu dává uživatelům možnost si konverzační část přizpůsobit naprosto ke své libosti, na druhou stranu však působí problém, protože konverzace je renderována jako statické HTML a jednou vygenerovaný obsah lze velmi těžko změnit. Jeden z konkrétních příkladů, kde jsme s tímto narazili, je indikace, že kontakt, se kterým chatujeme, právě píše. Můj návrh při implementaci této funkce byl přidat indikaci přímo do pohledu konverzace a odebrat ji, jakmile kontakt přestane psát. To by vyžadovalo přímou ma-

nipulaci HTML kódu, což by implementaci značně ztížilo. Jako druhý příklad lze uvést indikaci doručené zprávy, např. přidáním značky před zprávu. Řešení by opět vyžadovalo přímou manipulaci vygenerovaného HTML (z C++ rozhraní), přičemž řešení musí fungovat ve všech možných stylech.

`AdiumThemeView` svými možnostmi docela limituje potenciál konverzačního pohledu, navíc představuje poměrně velkou zátěž na údržbu a případné modifikace. Plánujeme ho tedy v některé z příštích verzí nahradit za pohled založený na QML, který nám umožní jednak větší flexibilitu a jednak větší kontrolu nad vlastním obsahem.

Třída `ChatWidget` obsluhuje textový kanál, není ale přímo klientem v architektuře Telepathy, tím je nadřazená aplikace, která `ChatWidget` integruje a třídě `ChatWidget` pouze předá získaný kanál. `ChatWidget` se tedy stará o vlastní zpracování příchozích zpráv, o odesílání zpráv a také o situaci, kdy je kanál uzavřen či zneplatněn (např. při přerušení spojení). Příchozí zprávy jsou přidány do pohledu, nejsou ale potvrzeny, dokud pohled není aktivován. Je-li okno s konverzačním pohledem aktivní, jsou zprávy potvrzeny ihned. Pokud je okno zavřeno dřív než jsou zprávy potvrzeny, zprávy zůstávají v kanálu a Channel Dispatcher kanál znovu deleguje schvalujícím klientům jako nový příchozí kanál. Při otevření kanálu je načtena poslední konverzace s daným kontaktem a do konverzačního pohledu jsou přidány standardně 4 poslední zprávy z poslední konverzace. Počet je možné změnit v nastavení.

V rámci projektu v programu Google Summer of Code 2012 vznikl také filtrovací framework pro `ChatWidget`, který na příchozí i odchozí zprávy aplikuje jednotlivé instalované (a povolené) filtry, které přímo modifikují tělo zprávy. Filtry pro příchozí zprávy jsou použity i na odchozí zprávy, které jsou přidávány do konverzačního pohledu, ale samozřejmě pouze lokálně. Filtry pro odchozí zprávy pak modifikují zprávu ještě před samotným odesláním.

Filtry jsou realizovány jako dynamicky načítatelné pluginy (každý plugin je sdílená knihovna exportována pomocí `KPluginFactory`), které spravuje `MessageProcessor`. Ten je realizován jako singleton. Filtry také mohou mít definovanou váhu, která určuje jejich prioritu při aplikaci na zprávu. Po načtení pluginů `MessageProcessorem` jsou pluginy podle této váhy seřazeny a v tomto pořadí pak aplikovány.

Mezi výchozími filtry je např. zobrazování náhledu obrázků, na které uživatel obdržel ve zprávě odkaz. Ve zprávě se detekují odkazy končící na přípony známých obrazových formátů a protože konverzační pohled je HTML, vloží se před zobrazením zprávy na její konec HTML značka `<img src="%1" />`, kde parametrem src je detekovaný odkaz na obrázek. Podobně funguje YouTube filtr, který při přijetí odkazu na YouTube video do pohledu vkládá samotné video. Dalšími filtry jsou náhrada textových „smajlíků“ za obrázky z přednastavené sady, formátování textu podle jednoduchých značek (/kurzíva/, _podtržení_, *tučně*, -přeškrtnuto-), čtení příchozích zpráv pomocí systému Text-To-Speech a využívání přednastavených webových zkratek (např. „gg:telepathy“ bude nahrazeno za „`http://www.google.com/search?q=telepathy&ie=UTF-8&oe=UTF-8`“; přednastavených je asi 150 zkratek a uživatel si může definovat své vlastní). I notifikace o příchozích zprávách jsou realizovány jako filtr. Speciálním filtrem především pro vý-

vojáře je Bugzilla filtr, který v textu hledá regulérní výraz `BUG:[ ]*(\\d+)`, pokud je nalezen, je z předdefinované Bugzilly (výchozí je bugs.kde.org) získán titulek a stav chyby a ty jsou připojeny ke zprávě. Stačí tedy odeslat např. „bug: 307326“ a oběma stranám se ve zprávě zobrazí „[BUG 307326] Avatars of aMSN users not displayed RESOLVED (WONTFIX)“.

`ChatWidget` obsluhuje také kanály pro více-uživatelský chat. V tomto případě je použit odlišný styl od běžných chatů, který více vyhovuje více-uživatelským chatům.

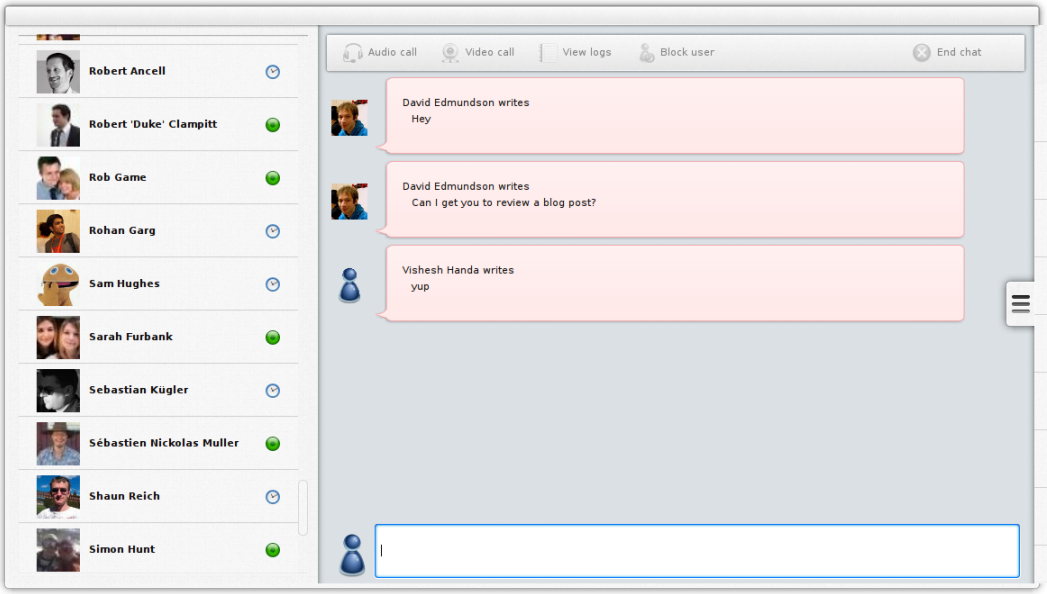
Telepathy klientem, kterému Channel Dispatcher deleguje kanály, je aplikace Text UI, která má záložkové rozhraní (tabs) a pro každý nový kanál vytváří novou záložku. Každá záložka pak obsahuje vlastní `ChatWidget`. Popisek záložky je jméno kontaktu a mění barvu podle stavu kanálu - kontakt píše zprávu, text je zelený; kanál obsahuje nepřečtené zprávy, text je fialový; kanál byl uzavřen, text je šedý. Podobně se mění i ikona záložek, která standardně ukazuje status kontaktu a mění se v případě, že kontakt píše nebo nám kontakt zaslal zprávu. Popisek a ikona aktivní záložky jsou přeneseny jako popisek a ikona hlavního okna aplikace. Hlavní okno dále obsahuje hlavní menu a konfigurovatelnou nástrojovou lištu, na které jsou ve výchozím stavu tlačítka pro zahájení audio/video hovoru, zaslání souboru a vyhledávání v aktivní konverzaci. Z hlavního menu aplikace je pak možné spustit modul s nastavením Text UI. Nastavení poskytuje možnost změnu stylu konverzačního pohledu pro normální i více-uživatelské chaty, možnost mít pro každý chat vlastní okno, nastavení počtu zpráv z historie, které se se zahájením chatu zobrazí a nastavení jednotlivých filtrů.

Pokud uživatel ztratí spojení např. výpadkem síťového spojení či prostým odpojením účtů, všechny otevřené kanály (aktivní chaty) zanikají (jsou vázány na spojení). Po opětovném navázání spojení projde aplikace otevřené záložky a obnoví všechny předešlé kanály.

3.6 Budoucnost

Část KDE vývojářů se nyní zaměřuje na vývoj nového pracovního prostředí pod názvem Active, založeného na technologii Plasma a zaměřeného především na mobilní zařízení s dotykovým ovládáním. Jedním z našich budoucích projektů bude právě uživatelské rozhraní pro Active, které bude dobře použitelné s dotykovým ovládáním. Rozhraní bude celé v QML a bude spojovat seznam kontaktů a textový chat v jednom okně, přičemž v seznamu kontaktů bude skupina „Aktivní konverzace“ a ta bude sloužit jako záložkové rozhraní pro chat. Tento projekt je zatím ve stádiu návrhu s prototypem uživatelského rozhraní v QML.

Zbytek projektu čeká v blízké budoucnosti několik portací - z frameworku Qt4 na novější Qt 5 a z KDELibs na KDE Frameworks 5. Tyto portace by neměly být příliš náročné, protože Qt 5 i KDE Frameworks 5 se snaží zachovat maximální API kompatibilitu. Třetí portací bude přechod na centrální správu účtů v prostředích KDE Plasma. Aktuálně si každá aplikace, která poskytuje jakoukoliv integraci se vzdálenými účty, řeší autentizaci po svém. S KDE Software Compilation 4.11 by měl přijít nový systém, který poskytuje „autentizaci účtů jako službu“, tedy stačí do systému přidat účet pouze jednou na jednom místě a všechny aplikace budou mít přístup k autentizačním údajům. Podobný systém již



Obrázek 12: Návrh uživatelského rozhraní pro Plasma Active

funguje také v GNOME 3 a Unity. Systém v KDE SC bude založen na stejné technologii, jako ten v Unity a oba budou navzájem kompatibilní. Systém podporuje také integraci s Telepathy a pro tu budeme muset upravit některé naše komponenty.

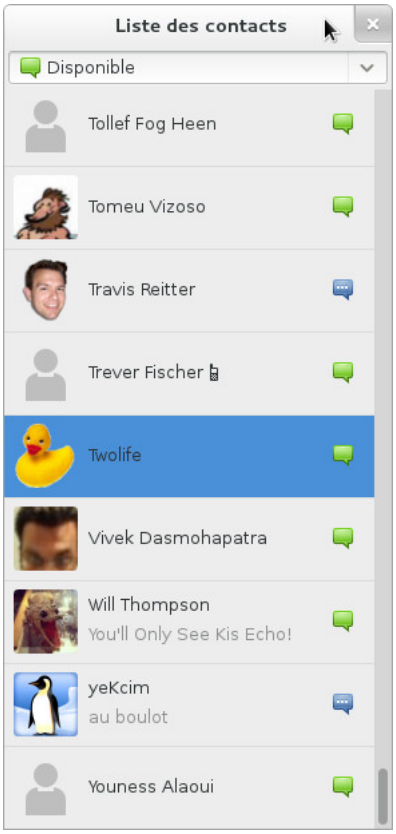
Velkým otazníkem do budoucna zůstává Telepathy-Qt. Přímému převzetí projektu bychom se rádi vyhnuli, protože by to přineslo novou a poměrně velkou zátěž na údržbu (najednou v projektu přibude více než 130 000 řádků kódu). Pokud ale nebude jiné východisko, jsme připraveni správcovství projektu převzít.

V době psaní této práce také začínají práce na integraci Nepomuku do projektu. Nepomuk je sémantické rozšíření pracovního prostředí a v KDE Telepathy bude využit k poskytování kontaktů a následně k tvorbě tzv. „metakontaktů“, což je sjednocení stejných kontaktů z různých účtů (např. máme na kolegu kontakt na ICQ i GTalku; osoba na druhé straně je stejná, liší se pouze použitý účet pro komunikaci). Mezi modely a pohledy tak přibude ještě jedna vrstva, která dá kontaktům sémantický rozměr. Díky tomu bude možné např. sémanticky označit soubor kontaktem, který nám jej poslal, přiřadit kontakt k fotce, na které je, nebo zobrazit aktuální online status v e-mailovém klientovi. Tím postupně vznikne komplexní strom obsahující nejrůznější data, která máme s kontaktem spojeny. Toto sémantické úložiště kontaktů nebude pouze doménou KDE Telepathy, ale bude implementováno v celém pracovním prostředí, především pak v aplikacích spravujících naše osobní informace (PIM) jako e-mailový klient nebo kalendář. Realizace tohoto řešení je poměrně komplexní; vyžaduje precizní koordinaci všech zainteresovaných stran a především pak připravenost cílených aplikací.

Mezitím se projekt bude přibližovat k verzi 1.0, která bude milník především pro vývojáře, neboť přinese stabilní API a ABI, potřebné dokumentace a naše sdílená knihovna už nebude pouze interní knihovnou, ale knihovnou dostupnou všem aplikacím.

4 Empathy

Aktuálně jedinou další existující implementací Telepathy klienta na desktopu je Empathy. Klient Empathy je založen na frameworku GTK, na kterém je vystavěno také pracovní prostředí GNOME. Vývoj klienta začal už v roce 2007 a od zhruba poloviny roku 2009 se pak stává přímo součástí GNOME a je spolu s ním také vydáván, převzato bylo také verzování. S příchodem GNOME 3 v dubnu 2011 se začíná pracovat na hlubší integraci klienta v pracovním prostředí. Empathy byl původně vyvíjen jako zcela monolitický klient, který v jednom procesu implementoval všechnu funkcionalitu (seznam kontaktů, autentizační, schvalovací i chatovací klient). Prvním krokem k lepší integraci tedy byla větší modulárnost.



Obrázek 13: Aktuální podoba Empathy v prostředí GNOME 3 (zdroj: http://blog.desmottes.be/)

V době psaní práce se Empathy architekturou blíží více architektuře KDE Telepathy - jednotlivé komponenty jako chatovací klient, klient pro audio/video hovory nebo autentizační klient, již běží ve vlastním procesu. V prostředí GNOME 3 je schvalujícím klientem podobně jako v KDE samotné pracovní prostředí, aplikace pro seznam kontaktů má ale i nadále svého vlastního schvalujícího klienta (což je v architektuře Telepathy naprosto v pořádku). To usnadňuje použití Empathy i mimo GNOME 3 prostředí. Aplikace pro

seznam kontaktů běží po prvním spuštění neustále, zavření okna aplikace způsobí pouze schování uživatelského rozhraní, samotný proces běží dál. Tento „workaround“ byl zaveden, protože aplikace startuje poměrně pomalu. Pokud uživatel aplikaci spustí a existuje již běžící proces, pouze se vyvolá zobrazení uživatelského rozhraní a uživatel má dojem, že aplikace startuje nesmírně rychle. Stejný postup jsme zvažovali i v naší aplikaci pro seznam kontaktů, nakonec jsme jej ale zavrhli, protože naše aplikace startuje do plně použitelného stavu velmi rychle (je sice vidět prodleva mezi otevřením a zobrazením kontaktů, je ale přijatelně malá; stoupá však lineárně s počtem kontaktů uživatele).

Empathy jako svého výchozího IM klienta nasadila také distribuce Ubuntu, která dříve poskytovala prostředí GNOME 2, pak se ale vydala cestou vlastního, na GNOME založeného prostředí Unity, které je stejně jako GNOME postaveno nad GTK. Empathy tedy zůstal výchozím klientem i pro Unity. Nyní se Unity přesouvá z GTK na Qt a existuje jistá šance, že Empathy vystřídá náš klient KDE Telepathy, žádné konkrétní plány nám ale nejsou známy.

GNOME 3.2 přineslo instantní komunikaci skutečně integrovanou do pracovního prostředí. Při obdržení nové zprávy je při spodní hraně obrazovky zobrazena malá notifikace, která se najetím myši roztáhne a zobrazí textové pole pro možnost okamžité odpovědi, bez nutnosti otevírat vlastní okno chatu. Podobně fungují také příchozí volání a příchozí přenos souborů. Poku je chat zahájen samotným uživatelem, je standardně otevřeno klasické okno chatu, které stejně jako KDE Telepathy využívá Adium stylů.

Seznam kontaktů byl od verze 3.6 vzhledově mírně upraven, aby vypadal podobně jako aplikace Kontakty, která spravuje všechny uživatelovy kontakty (včetně e-mailových atd.). Obě aplikace kladou velký důraz na avatary uživatele, neboť identifikace kontaktů podle obrázku je pro lidský mozek jednodušší než identifikace podle textového popisku. Na druhou stranu příliš velké ikony kontaktů dovolují v okně zobrazit maximálně na 10 kontaktů (samozřejmě záleží na rozlišení monitoru). To Empathy kompenzuje okamžitým vyhledáváním při psaní, je-li aplikace aktivním oknem. Podobně jako náš seznam kontaktů, i Empathy má ovládání statusu v okně seznamu aplikace. Nenabízí status „Nedostupný“ a použitý combobox je vždy textové pole, rozbalit se dá pouze šipkou na straně. Veškeré nastavení, jako zobrazení odpojených uživatelů či zobrazení skupin, je skryto v dialogu s preferencemi. Ve výchozím nastavení nezobrazuje Empathy skupiny. Oproti KDE Telepathy má Empathy také „Nejoblíbenější “ kontakty, které poskytuje sledovací systém Zeitgeist. Stejnou funkci máme v plánu i pro náš seznam kontaktů a zvažujeme také využít Zeitgeist.

Prostředí GNOME poskytuje pouze dva stavy v ovládání statusu - „Dostupný“ a „Odpojen“. Také Empathy podporuje automatické nastavení statusu po nečinnosti uživatele. Pokud Empathy běží v GNOME 3, o nastavení statusu se stará vlastní prostředí, které nastaví přímo status „Nedostupný“ (přeskakuje se „Pryč“) spolu s aktivací spořiče obrazovky. Mimo GNOME 3 má Empathy vlastní detekci nečinnosti uživatele, která nejprve nastaví „Pryč“ a po dalším intervalu pak „Nedostupný“. GNOME 3 má také systémový koncept „Zaneprázdněn“, který se aktivuje spuštěním aplikace na celou obrazovku (např. video přehrávač). Systém nastaví status „Zaneprázdněn“ také v Empathy.

Protože je Empathy starší klient než KDE Telepathy a navíc vyvíjen stejnými lidmi,

kteří pracují na samotném projektu Telepathy, často používáme Empathy jako referenční implementaci a nezřídka konzultujeme také zdrojový kód.

5 Jak probíhá vývoj

Vývoj celého projektu je naprosto otevřený a přispět či zapojit se do něj může kdokoli. Kód je uložen ve veřejných git repositářích hostovaných na serverech KDE. Procházet a klonovat („git clone“) repositář může každý, pro práva zápisu do repositáře je nutné mít aktivní účet KDE přispěvatele. Ten zahrnuje práva pro zápis do všech git i svn repositářů na serverech KDE. Práva zápisu do všech repositářů zajišťuje Manifest KDE, který vznikl za účelem jasné definice, co je a co není KDE projekt, přičemž každý KDE přispěvatel má práva zápisu do každého KDE projektu [8]. Pro získání účtu s právy zápisu je potřeba, aby se alespoň jeden stávající KDE vývojář za žadatele zaručil a jeho žádost potvrdil. Povinnosti ručitele je prvních pár týdnů dohlížet na commity žadatele. Mohlo by se zdát, že takto otevřený model je dost nebezpečný, protože kdokoli může cokoliv změnit, historie KDE serverů však tuto domněnku nepodporuje. Za dlouhá léta provozu nebyl ale zaznamenán žádný incident v podobě neoprávněných změn za účelem poškození.

V KDE je velmi dobrým zvykem před provedením změn v kódu nejprve požádat o revizi provedených změn. K tomu slouží nástroj Review Board - webová aplikace umožňující pohodlnou kolaborativní revizi změn, dostupná zcela zdarma pod MIT licencí. Princip fungování je velmi jednoduchý - autor změn nahraje patch, přidá vysvětlující popis a skupinu či konkrétní osoby, které o revizi žádá. Skupina anebo osoby poté obdrží email s žádostí. Díky plné integraci s git repositáři dokáže Review Board zobrazit nejen samotný patch, ale i zbytek změněných souboru, což se velmi hodí, pokud potřebujeme rychle náhled do zbylé části souboru. Vývojáři provádějící revizi pak mají možnost jednotlivé části kódu komentovat, případně komentovat již existující komentáře. Pro odsouhlasení změn má Review Board funkci „Ship It!“, kterou vývojář označí patch jako schválený a přebírá tak za něj částečnou zodpovědnost. Otevřené a schválené žádosti o revize lze automaticky uzavřít spolu se zapsáním commitů do repositáře a to přidáním klíčového slova „REVIEW: 123456“, kde „123456“ je samozřejmě číslo žádosti v Review Board.

K provádění rychlých revizí je velmi často využíván jednoduchý sdílecí prostor nazývaný „Pastebin“, kam lze vložit jakýkoliv text a obratem dostaneme odkaz na stránku se sdíleným textem, který bude po určité době automaticky smazán. KDE hostuje vlastní Pastebin server s veřejným HTTP API, díky kterému lze sdílení patchů v kombinaci s jednoduchým skriptem zjednodušit na `git diff  pastebinit`—. Výstupem je vygenerovaný odkaz, který lze sdílet s dalšími vývojáři, avšak vzhledem k povaze služby Pastebin („paste and forget“) je předpokladem sdílení reakce v horizontu minut; po delší době (standardně 30 minut) je sdílený text odstraněn. Pro menší patche je Pastebin rychlejší a pohodlnější volbou než vytváření celé žádosti v Review Board. V případě schválení změn mimo Review Board je v KDE zažitým zvykem uvést jméno revidujícího vývojáře do popisu commitu.

V KDE Telepathy je revidováno více než 90% veškerých změn před zapsáním na server, a to i změny vedoucích vývojářů. Nikdo v týmu tedy „není víc“ než ostatní, všichni jsou si zcela rovni. Aktuálně má tým tři vedoucí vývojáře, tři další vývojáře udržující některé z komponent a několik dalších občasně přispívajících vývojářů. Vedoucí vývojáři,

aktuálně já, David Edmundson a Daniele Domenichelli, určují směr, kterým se projekt bude ubírat - at'už z pohledu architektury, uživatelského rozhraní či nových funkcí. Jeden z vedoucích vývojářů je pak pověřen přípravou vydání, která zahrnuje sestavení časového i funkčního plánu, vytvoření větví a značek v repositářích a vytvoření samotných distribuovatelných balíčků se zdrojovým kódem komponent, které jsou pak vydány.

Při přípravě vydání se nejprve stanoví přibližné datum vydání (měsíc), poté se sestaví seznam funkcí a oprav, které do vydání musí být zahrnuty. Na základě toho se zhruba odhadne doba potřebná k realizaci seznamu a zpřesní se časový plán. Ten pak obsahuje několik fází, při kterých se kladou omezení na vývoj. Zhruba měsíc před vydáním je ukončen vývoj veškerých nových funkcí a repositáře jsou zmrazeny pro nové funkce (tzv. „feature freeze"). Od této chvíle probíhají pouze opravy chyb a intenzivní testování. Poté je nutné ukončit přidávání nových přeložitelných řetězců a uvědomit překladatelské týmy o nadcházejícím vydání, aby tak měli šanci dokončit chybějící překlady. Zmražení repositářů pro nové přeložitelné řetezce („string freeze") je podle počtu nových či změněných řetězců dva až tři týdny před vydáním. Je-li objevena chyba v řetězci, vývojář pověřený vydáním může povolit výjimku a musí uvědomit tým překladatelů. 3 dny před datem vydání jsou repositáře označkovány („git tags") a z těchto značek pak vytvořeny distribuovatelné balíčky se zdrojovými kódy a překlady. Tyto balíčky jsou nahrány na FTP servery KDE a na příslušných kanálech je pak nové vydání oznámeno. Těmito kanály jsou obvykle The Dot (oficiální KDE zpravodajský server), Planet KDE (služba agregující blogy vývojářů; text oznámení je zde oproti The Dot více neformální). Nakonec je vydání oznámeno na mailing listu projektu KDE Telepathy.

Uživatelé jsou v oznámení o vydání vždy povzbuzováni, aby veškeré nalezené chyby hlásili do nástroje pro správu chyb - Bugzilly. KDE má jednu společnou instalaci Bugzilly pro všechny projekty, která se snaží být uživatelsky maximálně přívětivá. Při hlášení chyby vybere uživatel produkt, komponentu a vyplní požadované údaje. Každé nové hlášení je přidáno jako „nepotvrzené". Vývojáři pak hlášení prochází a jedná-li se o skutečnou chybu, hlášení potvrdí (přejde do stavu „potvrzené"), případně požádají o doplňující informace („chybí informace") nebo hlášení uzavřou („vyřešeno"), přičemž se uvádí i jak bylo hlášení vyřešeno (např. „neplatné hlášení", „již opraveno", „není naše chyba", „nebude se opravovat" atd.). Bugzilla je aktuálně předním místem pro získávání zpětné vazby uživatelů KDE Telepathy.

Ke koordinaci týmu a práce se používá nejčastěji právě mailing list (e-mailová konference), kde probíhá většina důležitých diskuzí i plánování. Mailing list je veřejně archivován a na členství nejsou žádné požadavky či omezení, přidat se může kdokoli. Dříve byly na tento mailing list zasílány i hlášení z Bugzilly, jak ale projekt rostl, rostl i počet hlášených chyb a s tím rostla i nepřehlednost mailing listu. Byl proto vytvořen speciální mailing list pouze pro Bugzillu. Druhým komunikačním nástrojem je veřejný IRC kanál #kde-telepathy na serveru Freenode, kde se velmi často řeší věci ad-hoc a ne vždy za přítomnosti všech členů týmu. IRC je také místem koordinačních meetingů. Kanál nemá striktní téma a velmi často se zde diskutuje o všem možném. Živé diskuze rádi podporujeme, náš tým tak vytváří otevřenější a přátelštější dojem.

Kromě online komunikace se celý tým také setkává alespoň jednou ročně na pár dní

osobně. Každý KDE projekt, který splňuje podmínky KDE Manifestu, má podle téhož Manifestu nárok na finanční podporu, která je zpravidla využívána právě pro zajištění setkání vývojářů (proplacení cestovních a ubytovacích nákladů). Finanční podpora je zajištěna neziskovou organizací KDE e.V., registrovanou v Německu. Kromě finanční podpory zajišťuje KDE e.V. např. také registraci a správu ochranných známek projektů či právní pomoc při ochraně licence projektů.

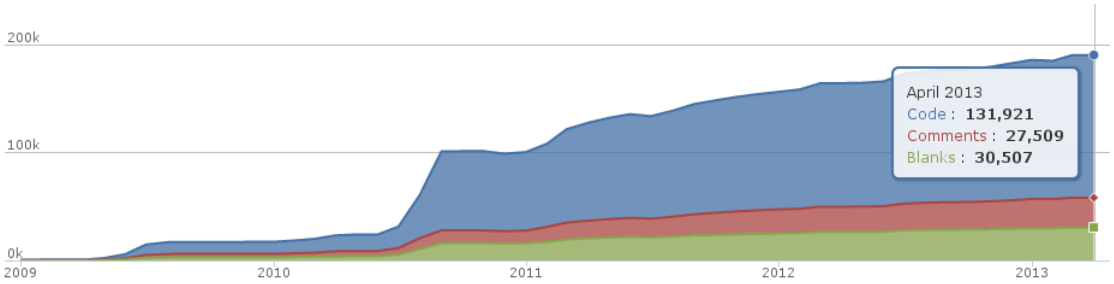
Osobní setkání, označované jako „kódovací sprinty“, jsou pro projekty obrovským přínosem. Obvykle trvají tři dny, kdy probíhá velmi intenzivní práce. Vývojáři bok po boku díky živým diskuzím udělají daleko více práce, než individuálně. Navíc se členové týmu, který je rozprostřen po celém světě, poznávají osobně (téměř bez výjimky jsou lidé schovaní za e-mailovými adresami naprosto odlišní od vytvořené představy z online komunikace), posilují se sociální vazby a vznikají nová přátelství, což obratem zefektivňuje týmovou spolupráci. Tyto sprinty posunují projekty po všech stránkách o velký kus vpřed.

Dalším místem setkávání a práce je každoroční sraz celé komunity KDE zvaný Akademy, který je vždy na jiném místě Evropy. První tři dny jsou ve znamení konference plné přednášek z nejrůznějších oblastí okolo KDE. Dalších pět dní je věnovaných aktivní a intenzivní práci s důrazem na spolupráci a koordinaci mezi různými projekty a jejich týmy. Akademy, jako jiné oblasti KDE, je veřejná událost plně otevřená všem lidem. KDE e.V. poskytuje i pro tuto akci finanční podporu, avšak žádosti o ni jsou posuzovány radou KDE e.V. individuálně, a protože finanční prostředky jsou omezeny, bohužel ne všem žádostem je vyhověno.

6 Statistiky

Následuje několik čísel o projektu. Zdrojem údajů je především sledovací služba Ohloh.net, která monitoruje všechny naše repositáře a vytváří z nich online dostupné statistiky. Ty se pochopitelně neustále mění, zde je zachycen stav koncem dubna 2013.

Od počátku projektu zaznamenaly repositáře celkem 7 188 comittů od 98 přispěvatelů, za posledních 12 měsíců to bylo 2 942 comittů od 46 přispěvatelů a za posledních 30 dní pak 254 comttů od 20 přispěvatelů. Dlouhodoběji se projekt drží na průměru 12 přispěvatelů měsíčně, což je velmi potěšující vzhledem k faktu, že základní tým tvoří 5 vývojářů.



Obrázek 14: Vývoj počtu řádků kódu (modře), komentářů (červeně) a volných řádků (zeleně) od počátku do dubna 2013

Nejaktivnější vývojář od počátku projektu je David Edmundson s aktuálně 1 134 commity. Druhý je Daniele E. Domenichelli s 946 commity a na třetí příčce jsem já s 765 commity.

Celkem bylo změněno 3 915 souborů, přidáno 539 442 řádků a odebráno 320 508 řádků kódu. Podle Ohlohu má projekt aktuálně celkem 131 921 řádků čistého kódu.

Nejvíce hlášení v Bugzille pochází od samotných vývojářů. Tento fakt je také ovlivněn tím, že pro každou chybu, kterou opravíme bez existujícího hlášení v Bugzille, nové hlášení před comittem nejprve vytvoříme. To nám velmi usnadňuje vytváření seznamů oprav mezi verzemi. Aktuálně je v naší Bugzille na 1 200 hlášení (chyby, pády i požadavky na funkce), již přes 1 000 hlášení je uzavřených a z toho zhruba 2/3 uzavřených jako „opraveno“. Zbylou třetinu tvoří duplicitní hlášení, hlášení neplatných chyb (např. vlastní vina uživatele atp.) či požadavky, které jsme zamítli. Duplicitní hlášení tvoří asi 8% všech hlášení. Nejvíce hlášení má komponenta Text UI, těsně za ní je seznam kontaktů a pak modul pro správu účtů. Většina hlášení má dva až čtyři komentáře, nejvíce komentářů je 33 a to hned u dvou hlášení.

7 Závěr

Po dvou letech práce je klient dodáván jako výchozí ve všech hlavních distribucích s prostředím KDE Plasma. Je přijmán velmi pozitivně a je neustále pod aktivním vývojem. Na každoročním setkání vývojářů Akademy v roce 2012 byl projekt nominován na cenu pro projekt roku, nakonec ocenění ale nezískal.

Práce na projektu KDE Telepathy byla pro mne obrovským přínosem po všech stránkách. Přinesla nová mezinárodní přátelství, výrazně prohloubila mé znalosti a zkušenosti s vývojem softwaru a managementem týmu a projektu jako takového. Díky své práci v KDE jsem také měl možnost cestovat na různá setkání v Evropě i USA, vyzkoušet si přednášení na mezinárodních konferencích a v neposlední řadě mě má nespokojenost se stavem instantní komunikace, která stála na počátku této práce, dovedla až na pracovní pozici vývojaře ve společnosti Blue Systems, která je dnes hlavním sponzorem KDE aktivit.

Přínos práce je pak především pro uživatele KDE prostředí, kteří opět mohou spokojeně využívat instantní komunikaci v moderním, jednoduchém a velmi dobře použitelném klientovi. Spokojení uživatelé jsou pak největší odměnou pro náš tým, protože víme, že to, co děláme, děláme dobře a že má cenu v tom pokračovat.

Martin Klapetek

8 Reference

[1] MADELEY, Danielle, *Telepathy. [online]. [cit. 2013-04-30]. Dostupné z: http://www.aosabook.org/en/telepathy.html*

[2] *Telepathy D-Bus Interface Specification. [online]. [cit. 2013-04-30]. Dostupné z: http://telepathy.freedesktop.org/spec/*

[3] SAINT-ANDRE, Peter *XEP-0174: Serverless Messaging. [online]. [cit. 2013-04-30]. Dostupné z: http://xmpp.org/extensions/xep-0174.html*

[4] Gamma, Helm, Johnson, and Vlissides *Design patterns: elements of reusable object-oriented software. Boston: Addison-Wesley, 1995, 395 s. ISBN 02-016-3361-2.*

[5] KLAPETEK, Martin et al. *KTp/NamingPolicy. [online]. [cit. 2013-04-30]. Dostupné z: http://community.kde.org/KTp/NamingPolicy*

[6] MCVITTIE, Simon *Implementing D-Bus clients with telepathy-glib (and telepathy-qt4). [online]. [cit. 2013-04-30]. Dostupné z: http://smcv.pseudorandom.co.uk/2009/05/tp-proxy/*

[7] *Farstream - Audio/Video Communications Framework. [online]. [cit. 2013-04-30]. Dostupné z: http://www.freedesktop.org/wiki/Software/Farstream*

[8] *The KDE Manifesto [online]. [cit. 2013-04-30]. Dostupné z: http://manifesto.kde.org*